Dalarna

A Simplistic Capability-Based Dynamic Language Design For Data Race Freedom

Kiko Fernandez Reyes[1] James Noble[2] Isaac Oscar Gariano[2]

Erin Greenwood-Thressman[2] Michael Homer[2] Tobias Wrigstad[1]

[1]Uppsala University [2]Victoria University of Wellington

Dalarna

A Simplistic Capability-Based Dynamic Language Design For Data Race Freedom

Kiko Fernandez Reyes[1] James Noble[2] Isaac Oscar Gariano[2]

Erin Greenwood-Thressman[2] Michael Homer[2] Tobias Wrigstad[1]

[1]Uppsala University [2]Victoria University of Wellington

Dalarna

A Simplistic Capability-Based Dynamic Language Design For Data Race Freedom

Kiko Fernandez Reyes[1] James Noble[2] Isaac Oscar Gariano[2]

Erin Greenwood-Thressman[2] Michael Homer[2] Tobias Wrigstad[1]

[1]Uppsala University [2]Victoria University of Wellington

Dalarna

A Simplistic Capability-Based Dynamic Language Design For Data Race Freedom

Kiko Fernandez Reyes[1] James Noble[2] Isaac Oscar Gariano[2]

Erin Greenwood-Thressman[2] Michael Homer[2] Tobias Wrigstad[1]

[1]Uppsala University [2]Victoria University of Wellington

Motivation

- Object-oriented dynamic languages in the rise [1]:

- All programming languages have concurrency constructs

Goroutines **Async & Await** **Task**

Web Workers **Promise & Future**

[1] Most Loved Language from Stackoverflow'20 Survey:
https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages-loved

Motivation

- Object-oriented dynamic languages in the rise [1]:

- All programming languages have concurrency constructs

Goroutines **Async & Await** **Task**

Web Workers **Promise & Future**

[1] Most Loved Language from Stackoverflow'20 Survey:
https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages-loved

Motivation

- Object-oriented dynamic languages in the rise [1]:

- All programming languages have concurrency constructs

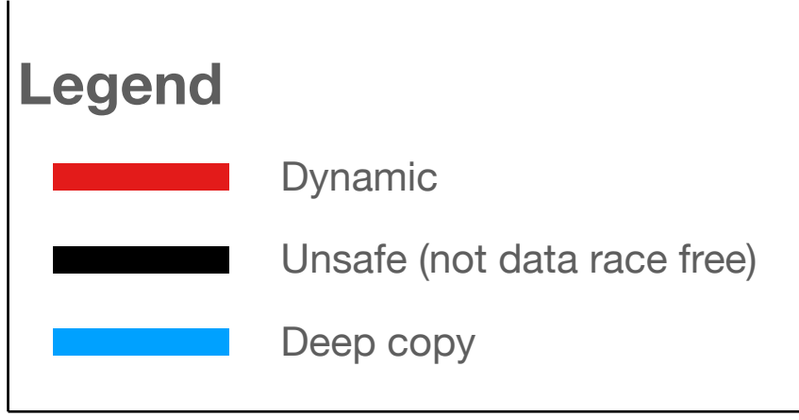
Goroutines **Async & Await** **Task**

Web Workers

How many are data race free?

[1] Most Loved Language from Stackoverflow'20 Survey:
https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages-loved

Motivation



- Object-oriented dynamic languages in the rise [1]:

- All programming languages have concurrency constructs

Goroutines **Async & Await** **Task**

Web Workers

How many are data race free?

[1] Most Loved Language from Stackoverflow'20 Survey:
https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages-loved

Legend

- Dynamic
- Unsafe (not data race free)
- Deep copy

Motivation

- Object-oriented dynamic languages in the rise [1]:

- All programming languages have concurrency constructs

Goroutines **Async & Await** **Task**

Web Workers

How many are data race free?

[1] Most Loved Language from Stackoverflow'20 Survey:
https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages-loved

Legend

- Dynamic
- Unsafe (not data race free)
- Deep copy

Motivation

- Object-oriented dynamic languages in the rise [1]:

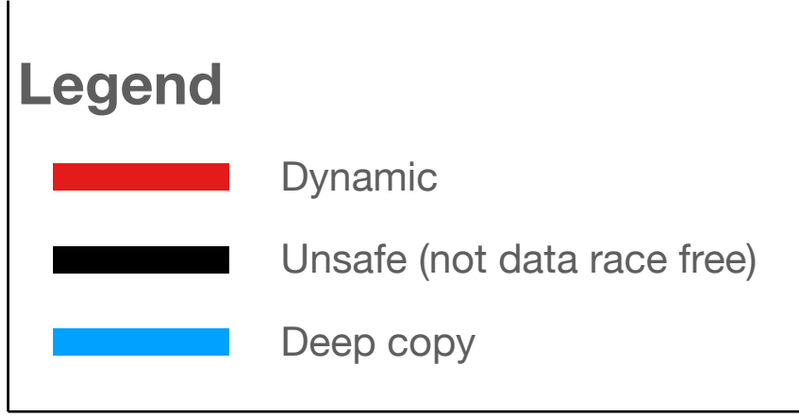
- All programming languages have concurrency constructs

Goroutines **Async & Await** **Task**

Web Workers

How many are data race free?

[1] Most Loved Language from Stackoverflow'20 Survey:
https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages-loved



Dalarna Programming Model

- Focused on *dynamic* languages
- Make programs *data race free* by mapping objects to *capabilities:*
  - Remove *data races* in 
  - Remove *deep copying* where possible 
- Embedded Dalarna into the Grace language

Dalarna Programming Model

$$
\begin{array}{rrcl}
\textit{Programs} & P & ::= & t \\
\textit{Objects} & O & ::= & K\ \mathtt{obj}\ \{\overline{F}\ \overline{M}\} \\
\textit{Fields} & F & ::= & f = w \\
\textit{Methods} & M & ::= & \mathtt{method}\ m(x)\ \{t\} \\
\textit{Terms} & t & ::= & w \mid \mathtt{let}\ x\ =\ e\ \mathtt{in}\ t \\
\textit{Expressions} & e & ::= & w \mid x.f \mid x.f = w \\
 & & \mid & x.m(w) \mid K\ \mathtt{copy}\ x \mid O \\
 & & \mid & \mathtt{spawn}\ (x)\ \{t\} \mid \blacksquare_i\ \iota \mid \upsilon \\
 & & \mid & x \leftarrow w \mid \ \leftarrow x \\
\textit{Variables} & w & ::= & x \mid \mathtt{consume}\ x \mid (K)\ w \\
\textit{Values} & \upsilon & ::= & \iota \mid \varnothing \mid \top \\
\textit{Capabilities} & K & ::= & \mathtt{imm} \mid \mathtt{local} \mid \mathtt{iso} \\
 & & \mid & \mathtt{unsafe} \mid K\ \&\ K'
\end{array}
$$

Dalarna Programming Model

Programs	P	::=	t
Objects	O	::=	$K\ \mathbf{obj}\ \{\overline{F}\ \overline{M}\}$
Fields	F	::=	$f = w$
Methods	M	::=	$\mathbf{method}\ m(x)\ \{t\}$
Terms	t	::=	$w \mid \mathbf{let}\ x\ =\ e\ \mathbf{in}\ t$
Expressions	e	::=	$w \mid x.f \mid x.f = w$
		$\mid$	$x.m(w) \mid K\ \mathbf{copy}\ x \mid O$
		$\mid$	$\mathbf{spawn}\ (x)\ \{t\} \mid \blacksquare_i\ \iota \mid \upsilon$
		$\mid$	$x \leftarrow w \mid \ \leftarrow x$
Variables	w	::=	$x \mid \mathbf{consume}\ x \mid (K)\ w$
Values	υ	::=	$\iota \mid \varnothing \mid \top$
Capabilities	K	::=	$\mathbf{imm} \mid \mathbf{local} \mid \mathbf{iso}$
		$\mid$	$\mathbf{unsafe} \mid K\ \&\ K'$

Dalarna Programming Model

$$
\begin{array}{rlll}
\textit{Programs} & P & ::= & t \\
\textit{Objects} & O & ::= & K\ \mathtt{obj}\ \{\overline{F}\ \overline{M}\} \\
\textit{Fields} & F & ::= & f = w \\
\textit{Methods} & M & ::= & \mathtt{method}\ m(x)\ \{t\} \\
\hline
\textit{Terms} & t & ::= & w \mid \mathtt{let}\ x\ =\ e\ \mathtt{in}\ t \\
\textit{Expressions} & e & ::= & w \mid x.f \mid x.f = w \\
 & & \mid & x.m(w) \mid K\ \mathtt{copy}\ x \mid O \\
 & & \mid & \mathtt{spawn}\ (x)\ \{t\} \mid \blacksquare_i\ \iota \mid \upsilon \\
 & & \mid & x \leftarrow w \mid \ \leftarrow x \\
\textit{Variables} & w & ::= & x \mid \mathtt{consume}\ x \mid (K)\ w \\
\textit{Values} & \upsilon & ::= & \iota \mid \varnothing \mid \top \\
\textit{Capabilities} & K & ::= & \mathtt{imm} \mid \mathtt{local} \mid \mathtt{iso} \\
 & & \mid & \mathtt{unsafe} \mid K\ \&\ K'
\end{array}
$$

Dalarna Programming Model

$$
\begin{array}{rrcl}
\textit{Programs} & P & ::= & t \\
\textit{Objects} & O & ::= & K\ \texttt{obj}\ \{\overline{F}\ \overline{M}\} \\
\textit{Fields} & F & ::= & f = w \\
\textit{Methods} & M & ::= & \texttt{method}\ m(x)\ \{t\} \\
\textit{Terms} & t & ::= & w \mid \texttt{let}\ x = e\ \texttt{in}\ t \\
\textit{Expressions} & e & ::= & w \mid x.f \mid x.f = w \\
 & & \mid & x.m(w) \mid K\ \texttt{copy}\ x \mid O \\
 & & \mid & \texttt{spawn}\ (x)\ \{t\} \mid \blacksquare_i\ \iota \mid \upsilon \\
 & & \mid & x \leftarrow w \mid \ \leftarrow x \\
\textit{Variables} & w & ::= & x \mid \texttt{consume}\ x \mid (K)\ w \\
\textit{Values} & \upsilon & ::= & \iota \mid \varnothing \mid \top \\
\textit{Capabilities} & K & ::= & \texttt{imm} \mid \texttt{local} \mid \texttt{iso} \\
 & & \mid & \texttt{unsafe} \mid K\ \&\ K'
\end{array}
$$

Dalarna Programming Model

$$
\begin{array}{rlll}
\textit{Programs} & P & ::= & t \\
\textit{Objects} & O & ::= & K\ \mathtt{obj}\ \{\overline{F}\ \overline{M}\} \\
\textit{Fields} & F & ::= & f = w \\
\textit{Methods} & M & ::= & \mathtt{method}\ m(x)\ \{t\} \\
\textit{Terms} & t & ::= & w \mid \mathtt{let}\ x = e\ \mathtt{in}\ t \\
\textit{Expressions} & e & ::= & w \mid x.f \mid x.f = w \\
 & & \mid & x.m(w) \mid K\ \mathtt{copy}\ x \mid O \\
 & & \mid & \mathtt{spawn}\ (x)\ \{t\} \mid \blacksquare_i\ \iota \mid \upsilon \\
 & & \mid & x \leftarrow w \mid \ \leftarrow x \\
\textit{Variables} & w & ::= & x \mid \mathtt{consume}\ x \mid (K)\ w \\
\textit{Values} & \upsilon & ::= & \iota \mid \varnothing \mid \top \\
\textit{Capabilities} & K & ::= & \mathtt{imm} \mid \mathtt{local} \mid \mathtt{iso} \\
 & & \mid & \mathtt{unsafe} \mid K\ \&\ K'
\end{array}
$$

Dalarna Programming Model

$$
\begin{array}{rrcl}
\textit{Programs} & P & ::= & t \\
\textit{Objects} & O & ::= & K\ \mathtt{obj}\ \{\overline{F}\ \overline{M}\} \\
\textit{Fields} & F & ::= & f = w \\
\textit{Methods} & M & ::= & \mathtt{method}\ m(x)\ \{t\} \\
\textit{Terms} & t & ::= & w \mid \mathtt{let}\ x\ =\ e\ \mathtt{in}\ t \\
\textit{Expressions} & e & ::= & w \mid x.f \mid x.f = w \\
 & & \mid & x.m(w) \mid K\ \mathtt{copy}\ x \mid O \\
 & & \mid & \mathtt{spawn}\ (x)\ \{t\} \mid \blacksquare_i\ \iota \mid v \\
 & & \mid & x \leftarrow w \mid \ \leftarrow x \\
\textit{Variables} & w & ::= & x \mid \mathtt{consume}\ x \mid (K)\ w \\
\textit{Values} & v & ::= & \iota \mid \varnothing \mid \top \\
\textit{Capabilities} & K & ::= & \mathtt{imm} \mid \ \mathtt{local} \mid \mathtt{iso} \\
 & & \mid & \mathtt{unsafe} \mid K\ \&\ K'
\end{array}
$$

Dalarna Programming Model

$$
\begin{array}{rlll}
\textit{Programs} & P & ::= & t \\
\textit{Objects} & O & ::= & K\ \mathtt{obj}\ \{\overline{F}\ \overline{M}\} \\
\textit{Fields} & F & ::= & f = w \\
\textit{Methods} & M & ::= & \mathtt{method}\ m(x)\ \{t\} \\
\textit{Terms} & t & ::= & w \mid \mathtt{let}\ x\ =\ e\ \mathtt{in}\ t \\
\textit{Expressions} & e & ::= & w \mid x.f \mid x.f = w \\
 & & \mid & x.m(w) \mid K\ \mathtt{copy}\ x \mid O \\
 & & \mid & \mathtt{spawn}\ (x)\ \{t\} \mid \blacksquare_i\ \iota \mid \upsilon \\
 & & \mid & x \leftarrow w \mid \ \leftarrow x \\
\textit{Variables} & w & ::= & x \mid \mathtt{consume}\ x \mid (K)\ w \\
\textit{Values} & \upsilon & ::= & \iota \mid \varnothing \mid \top \\
\textit{Capabilities} & K & ::= & \mathtt{imm} \mid \mathtt{local} \mid \mathtt{iso} \\
 & & \mid & \mathtt{unsafe} \mid K\ \&\ K'
\end{array}
$$

Dalarna Programming Model

$$
\begin{array}{rlll}
\textit{Programs} & P & ::= & t \\
\textit{Objects} & O & ::= & K\ \mathtt{obj}\ \{\overline{F}\ \overline{M}\} \\
\textit{Fields} & F & ::= & f = w \\
\textit{Methods} & M & ::= & \mathtt{method}\ m(x)\ \{t\} \\
\textit{Terms} & t & ::= & w \mid \mathtt{let}\ x\ =\ e\ \mathtt{in}\ t \\
\textit{Expressions} & e & ::= & w \mid x.f \mid x.f = w \\
 & & \mid & x.m(w) \mid K\ \mathtt{copy}\ x \mid O \\
 & & \mid & \mathtt{spawn}\ (x)\ \{t\} \mid \blacksquare_i\ \iota \mid v \\
 & & \mid & x \leftarrow w \mid \ \leftarrow x \\
\textit{Variables} & w & ::= & x \mid \mathtt{consume}\ x \mid (K)\ w \\
\textit{Values} & v & ::= & \iota \mid \varnothing \mid \top \\
\textit{Capabilities} & K & ::= & \mathtt{imm} \mid \mathtt{local} \mid \mathtt{iso} \\
 & & \mid & \mathtt{unsafe} \mid K\ \&\ K'
\end{array}
$$

Dalarna Programming Model

$$
\begin{array}{rlll}
\textit{Programs} & P & ::= & t \\
\textit{Objects} & O & ::= & K\ \mathtt{obj}\ \{\overline{F}\ \overline{M}\} \\
\textit{Fields} & F & ::= & f = w \\
\textit{Methods} & M & ::= & \mathtt{method}\ m(x)\ \{t\} \\
\textit{Terms} & t & ::= & w \mid \mathtt{let}\ x\ =\ e\ \mathtt{in}\ t \\
\textit{Expressions} & e & ::= & w \mid x.f \mid x.f = w \\
 & & \mid & x.m(w) \mid K\ \mathtt{copy}\ x \mid O \\
 & & \mid & \mathtt{spawn}\ (x)\ \{t\} \mid \blacksquare_i\ \iota \mid \upsilon \\
 & & \mid & x \leftarrow w \mid \ \leftarrow x \\
\textit{Variables} & w & ::= & x \mid \mathtt{consume}\ x \mid (K)\ w \\
\textit{Values} & \upsilon & ::= & \iota \mid \varnothing \mid \top \\
\textit{Capabilities} & K & ::= & \mathtt{imm} \mid \mathtt{local} \mid \mathtt{iso} \\
 & & \mid & \mathtt{unsafe} \mid K\ \&\ K'
\end{array}
$$

Dalarna Programming Model

- Capabilities annotations

	Effects			
Object	Read	Write	Alias	Transfer
Imm	●	×	●	●
Iso	●	●	×	●
Local	●	●	●	×
Unsafe	●	●	●	●

Dalarna Programming Model

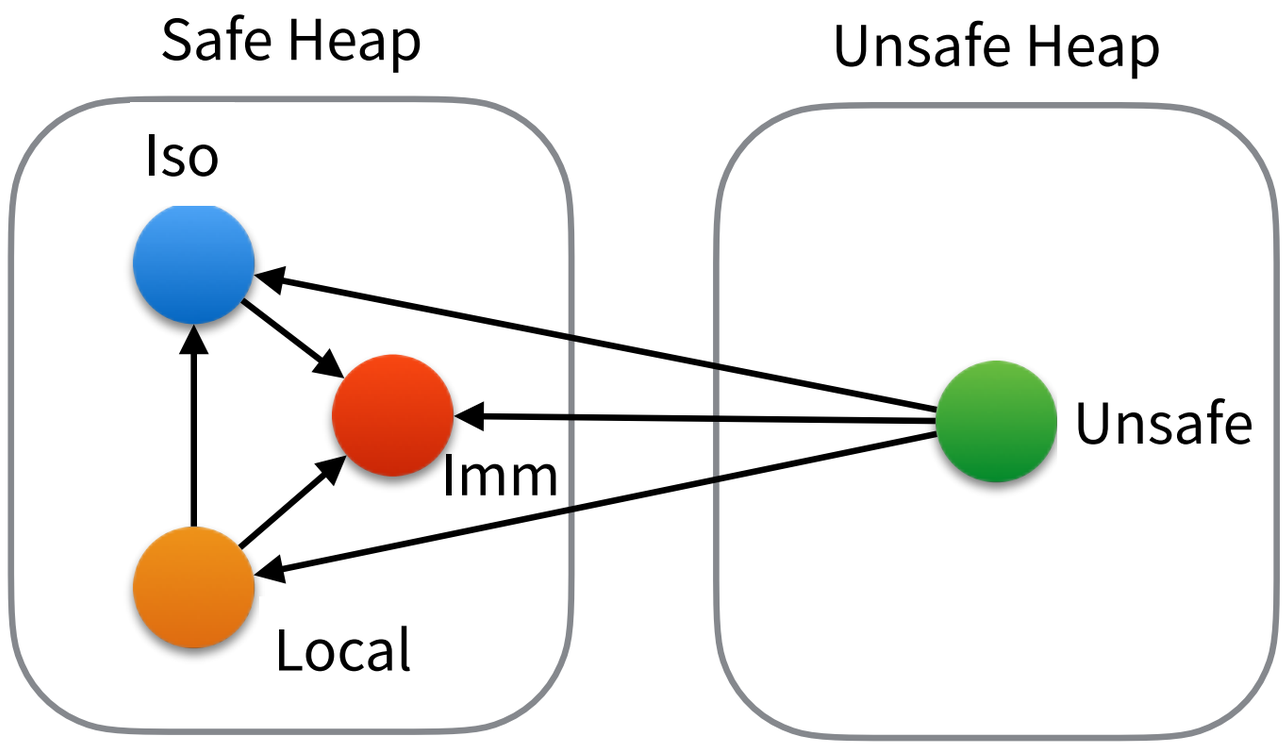
- Capabilities annotations

Object	Effects			
	Read	Write	Alias	Transfer
Imm	●	×	●	●
Iso	●	●	×	●
Local	●	●	●	×
Unsafe	●	●	●	●

Local & Imm = Read and Alias

Dalarna Programming Model

- Object capability containment relation



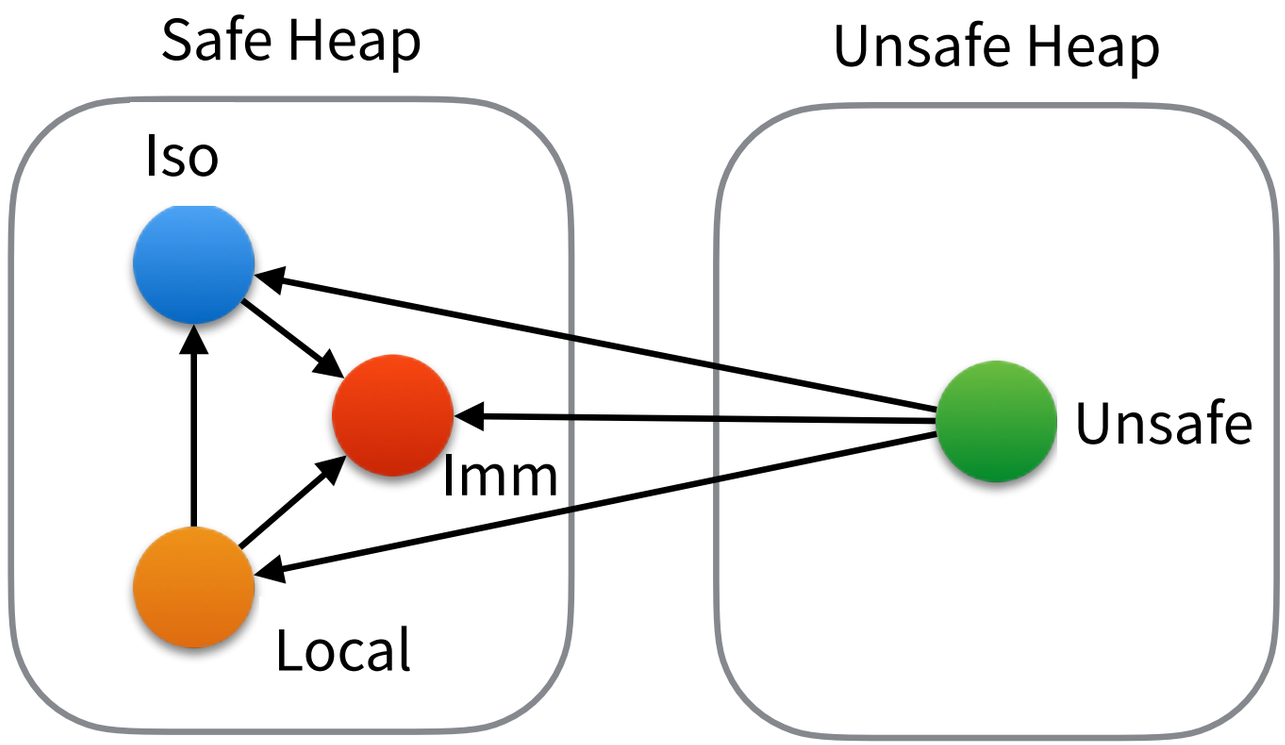
Dalarna Programming Model

- Object capability containment relation

This relation must **hold**:

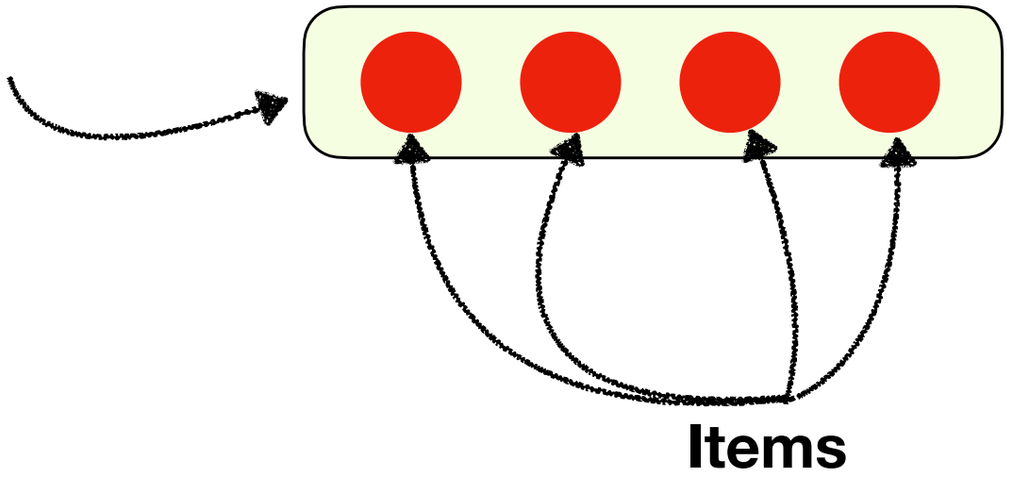
unsafe ≤ local ≤ iso ≤ imm

(Reflexive & Transitive)



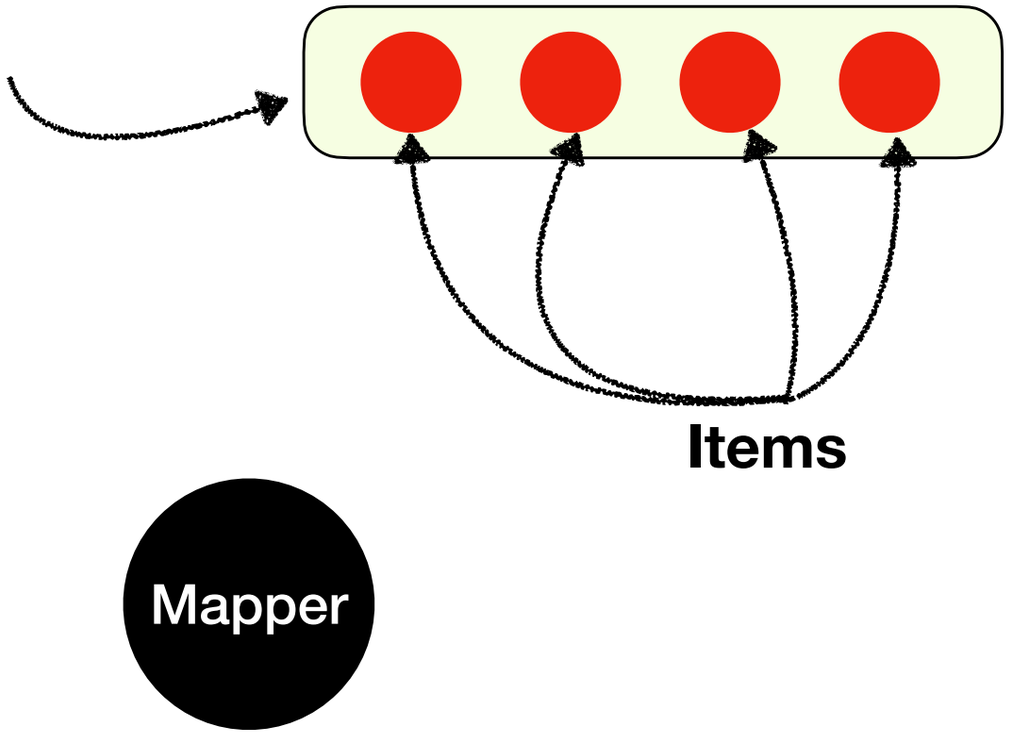
Dalarna Programming Model

Initial code



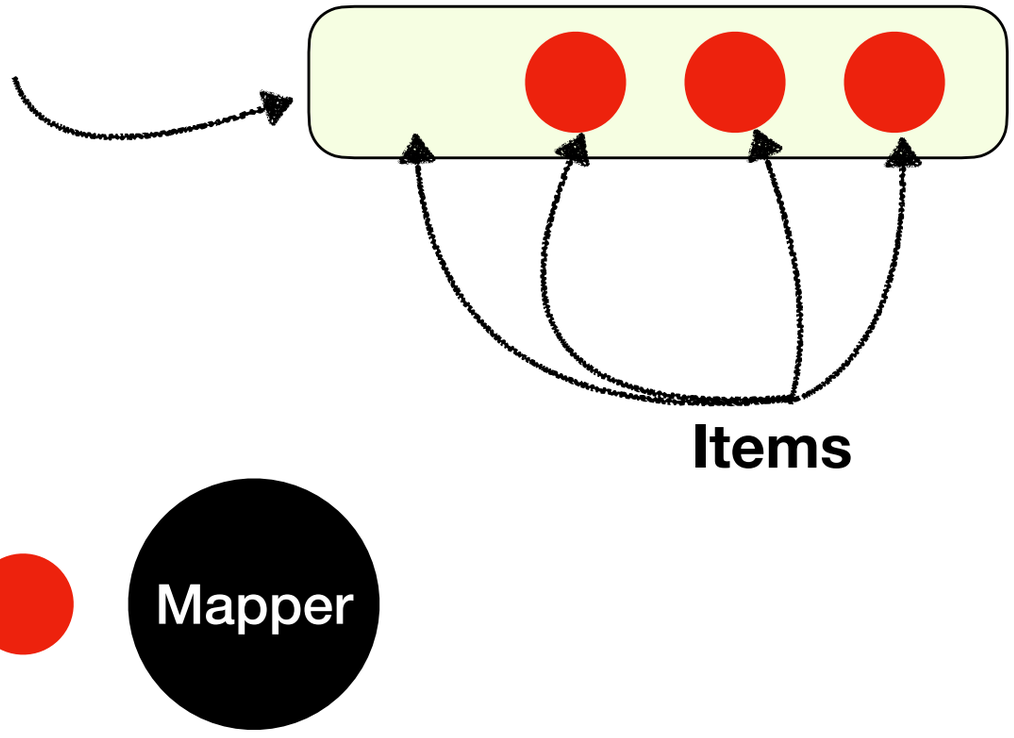
Dalarna Programming Model

Initial code



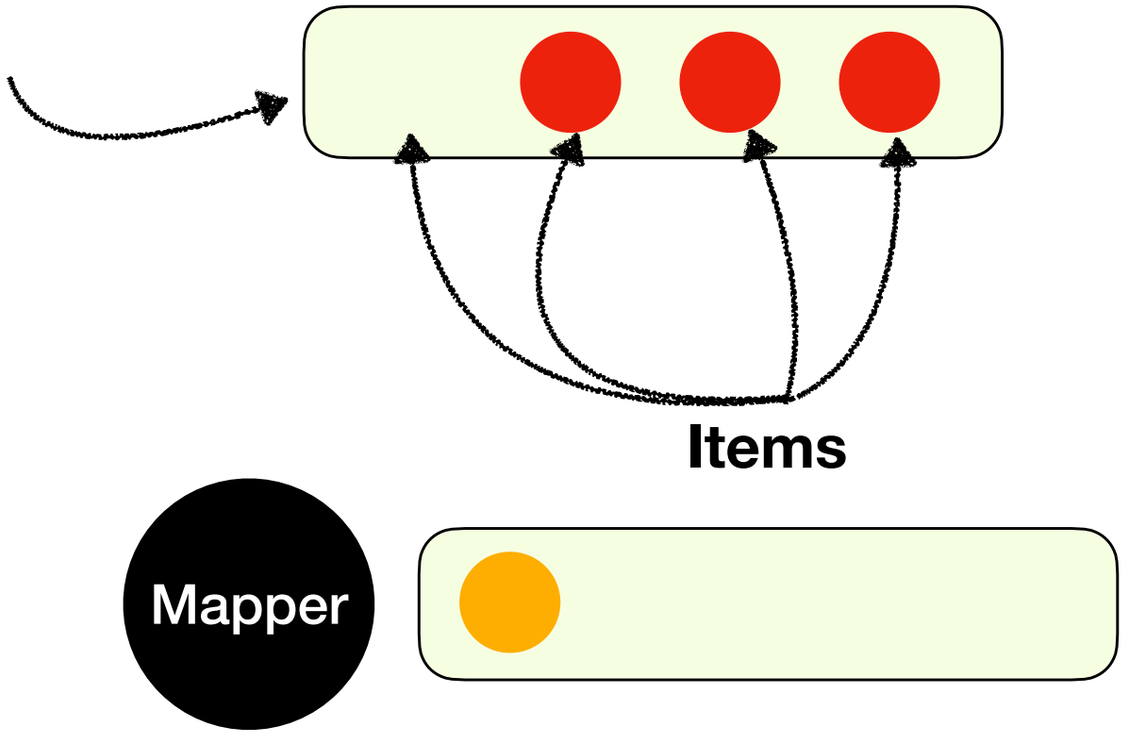
Dalarna Programming Model

Initial code



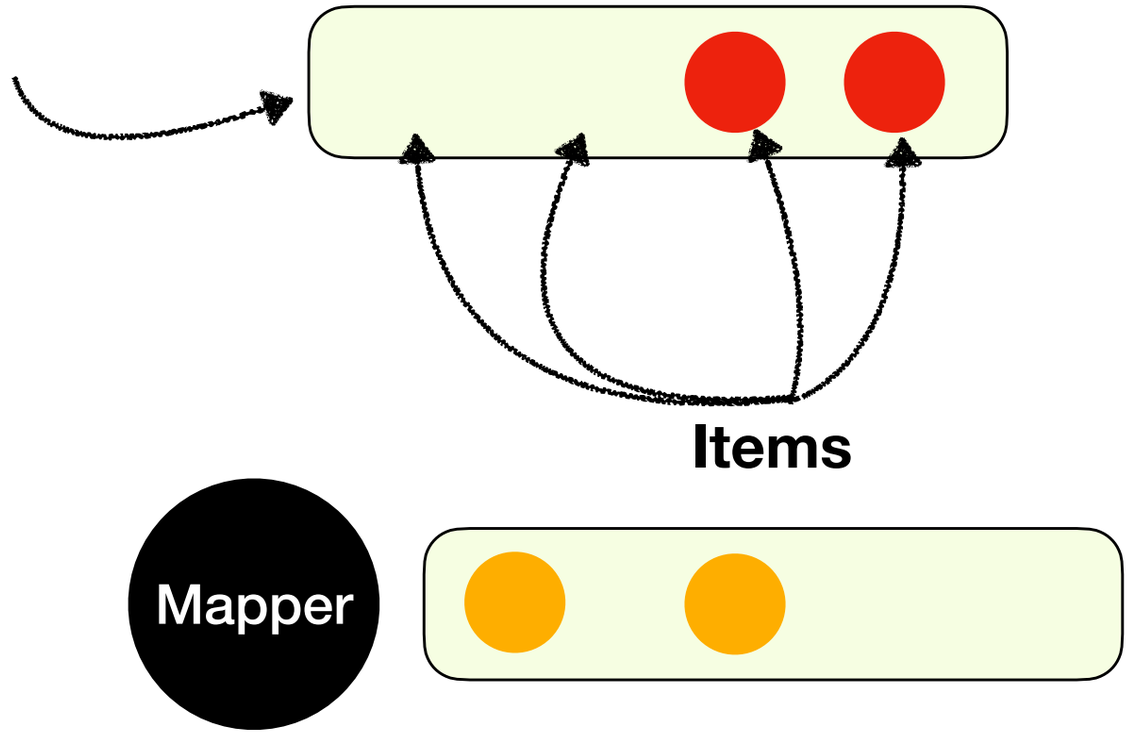
Dalarna Programming Model

Initial code



Dalarna Programming Model

Initial code



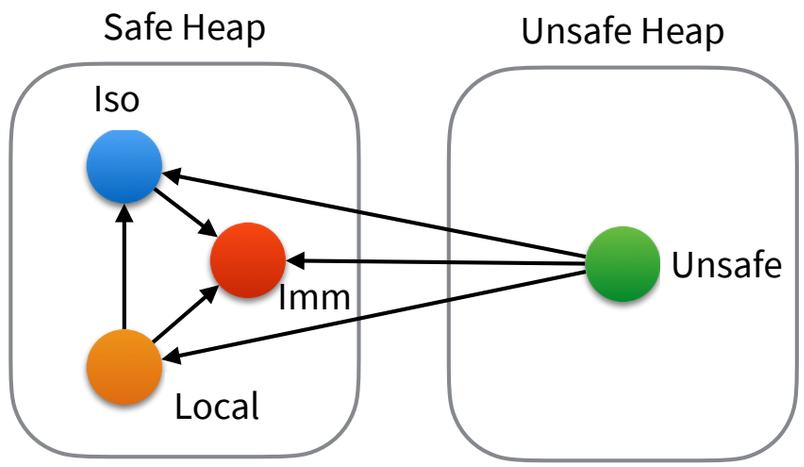
Dalarna Programming Model

Initial code

```
ls = object List {…}
ls.append(object 0 {…} )

mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
≡
ls = unsafe object List {…}
```



Dalarna Programming Model

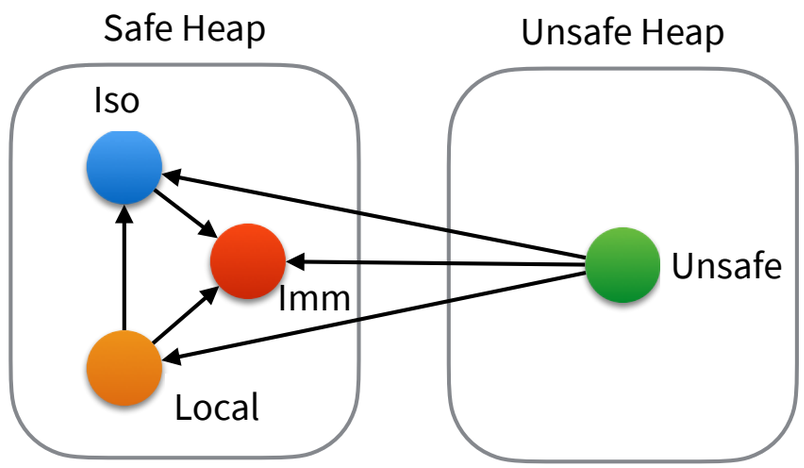
Initial code

```
ls = object List {…}
ls.append(object 0 {…} )
```

```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```



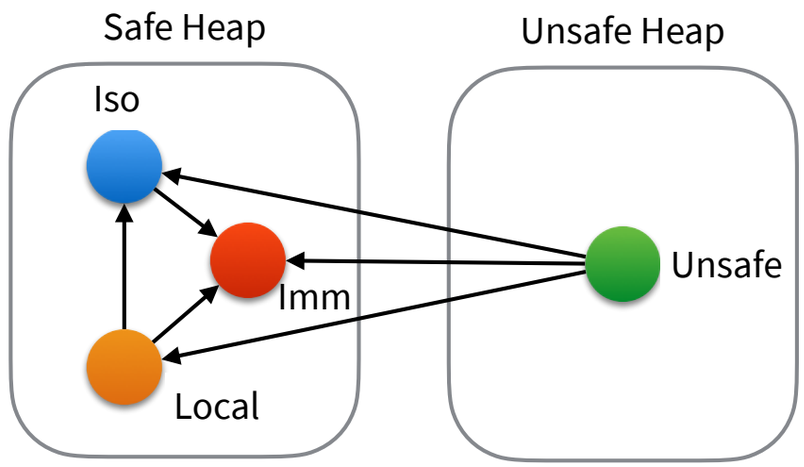
Dalarna Programming Model

Initial code

```
ls = object List {…}
ls.append(object 0 {…} )

mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
≡
ls = unsafe object List {…}
```



Dalarna Programming Model

Initial code

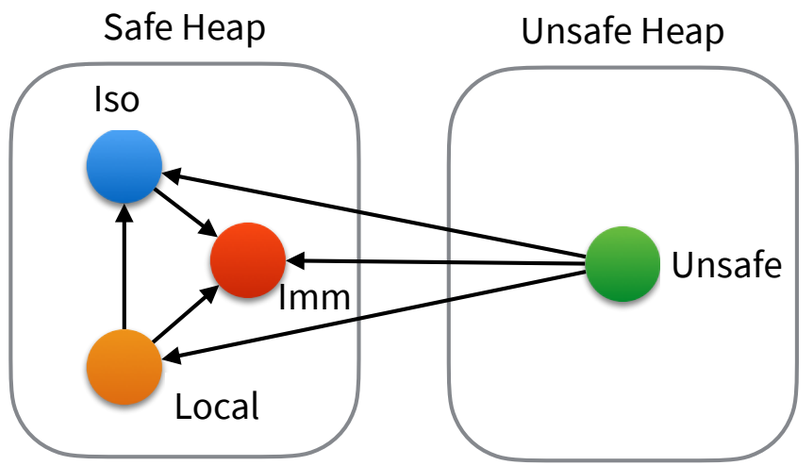
```
ls = object List {…}
ls.append(object 0 {…} )

mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
```

≡

```
ls = unsafe object List {…}
```



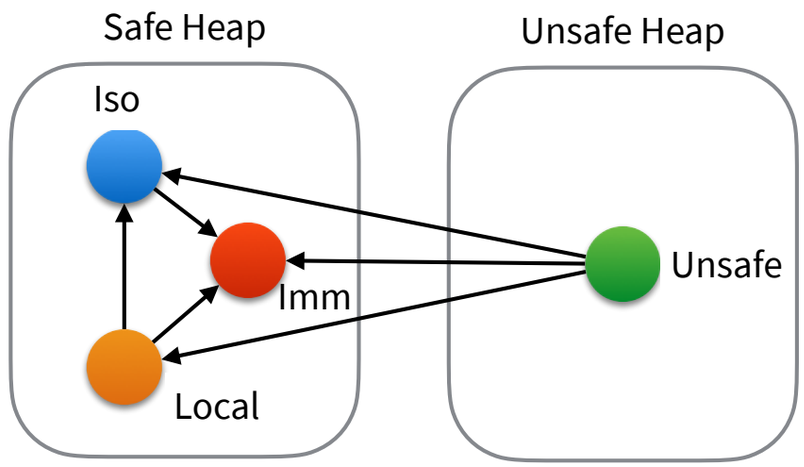
Dalarna Programming Model

Initial code

```
ls = object List {…}
ls.append(object 0 {…} )

mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
 ls = object List {…}
        ≡
ls = unsafe object List {…}
```



Dalarna Programming Model

Initial code

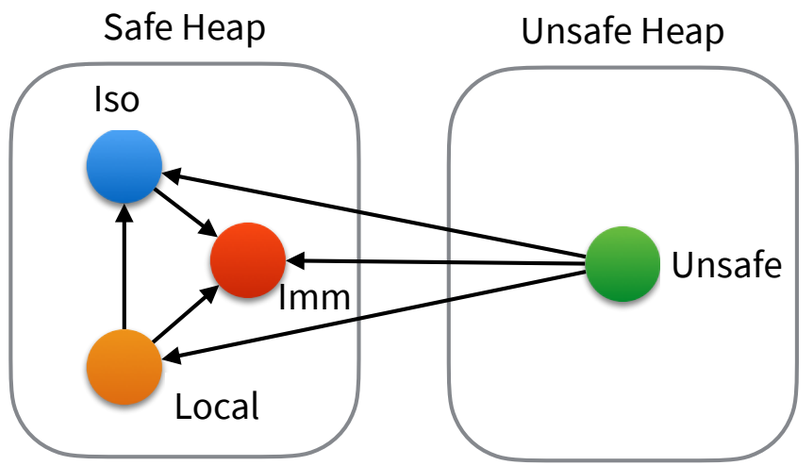
```
ls = object List {…}
ls.append(object 0 {…} )

mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
 ls = object List {…}
```

≡

```
ls = unsafe object List {…}
```



Dalarna Programming Model

Initial code

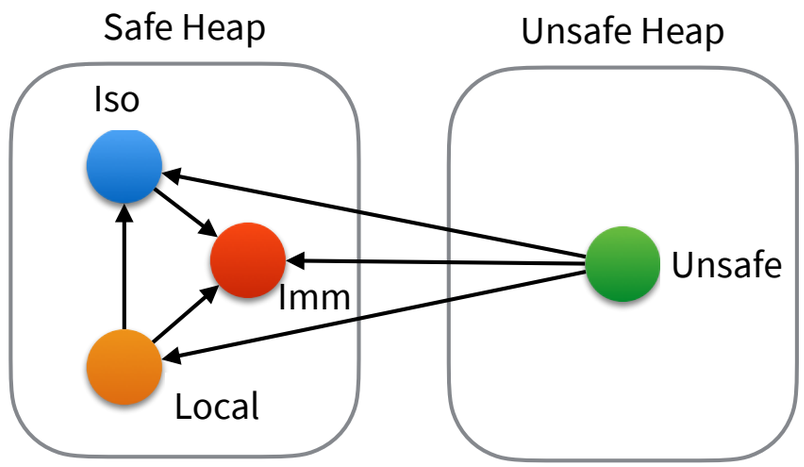
```
ls = object List {…}
ls.append(object 0 {…} )

mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
```

≡

```
ls = unsafe object List {…}
```



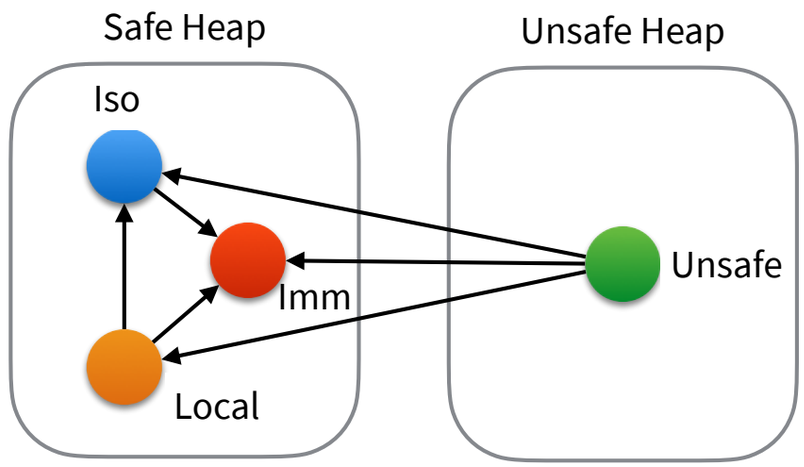
Dalarna Programming Model

Initial code

```
ls = object List {…}
ls.append(object 0 {…} )

mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
        ≡
ls = unsafe object List {…}
```



Dalarna Programming Model

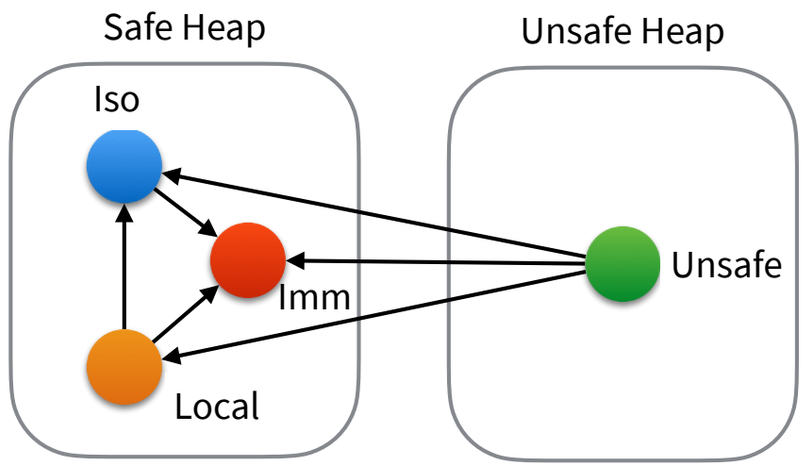
Initial code

```
ls = object List {…}
ls.append(object 0 {…} )

mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

This code is **not** data race free

Default object considered with **unsafe capability**

```
ls = object List {…}
≡
ls = unsafe object List {…}
```



Dalarna Programming Model

Initial code

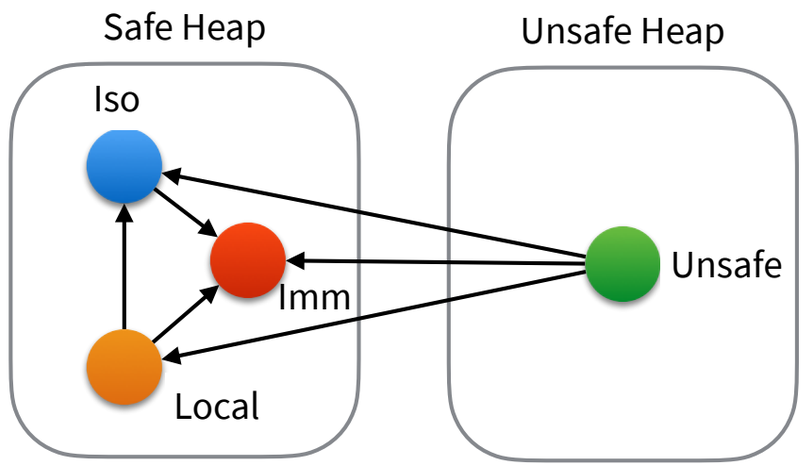
```
ls = object List {…}
ls.append(object 0 {…} )

mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

This code is **not** data race free

Do not share state,
but there is no enforcement

Default object considered with **unsafe capability**

```
ls = object List {…}
≡
ls = unsafe object List {…}
```



Dalarna Programming Model

~~Initial code~~ Isolated list

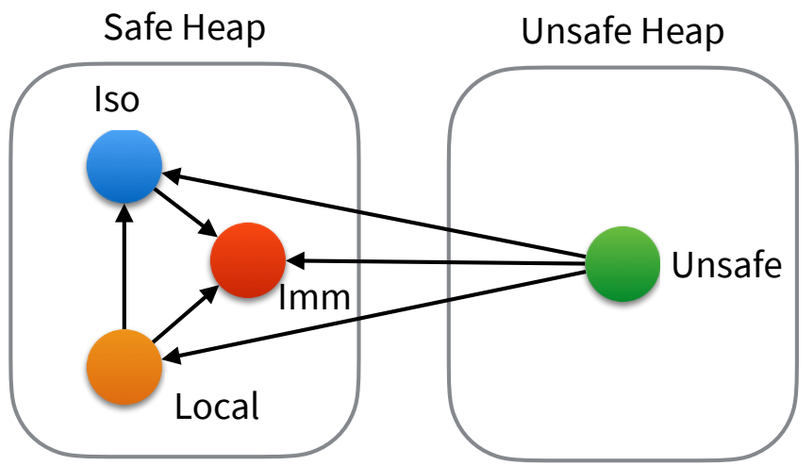
```
ls = iso object List {…}
ls.append(object 0 {…} )

mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
```

≡

```
ls = unsafe object List {…}
```



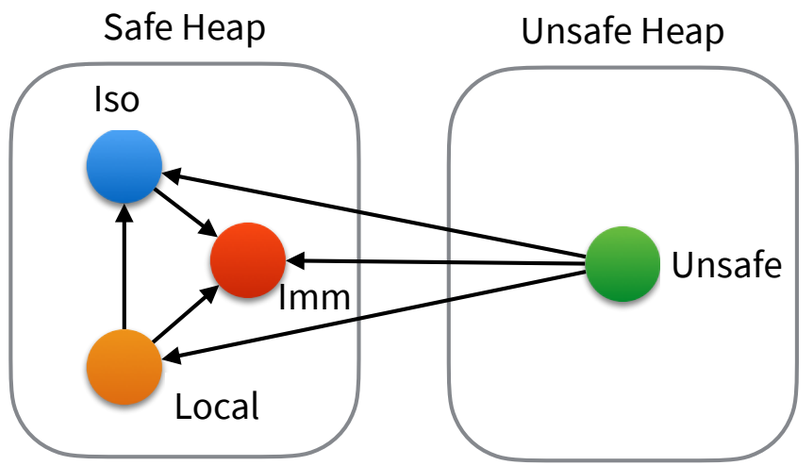
Dalarna Programming Model

~~Initial code~~ Isolate list

```
ls = iso object List {…}
ls.append(object 0 {…} )

mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
≡
ls = unsafe object List {…}
```



Dalarna Programming Model

~~Initial code~~ Isolate list

```
ls = iso object List {…}
ls.append(object 0 {…} )
```

Broken containment relation

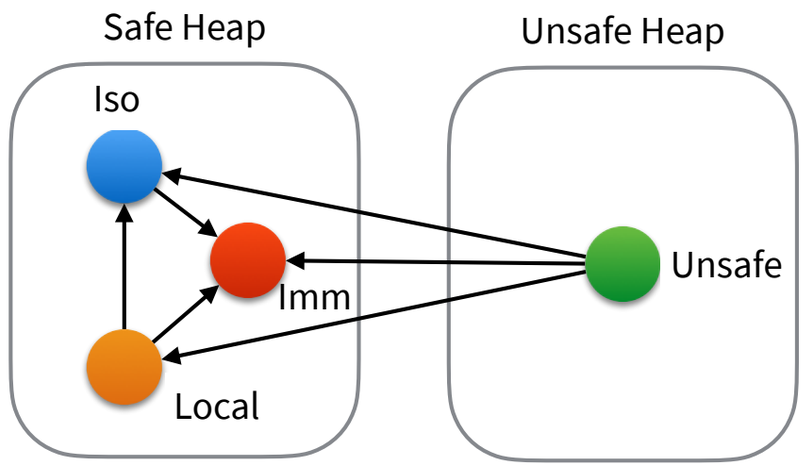
```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
```

≡

```
ls = unsafe object List {…}
```



Dalarna Programming Model

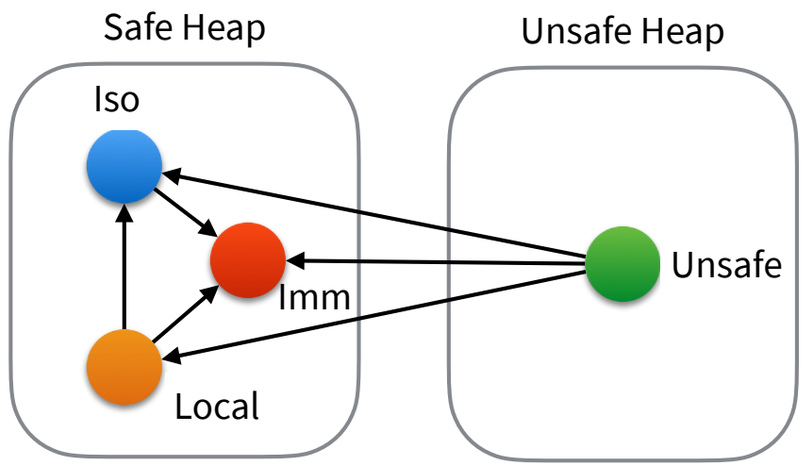
~~Initial code~~ Isolate list

```
ls = iso object List {…}
ls.append(imm object 0 {…} )
```

```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
≡
ls = unsafe object List {…}
```



Dalarna Programming Model

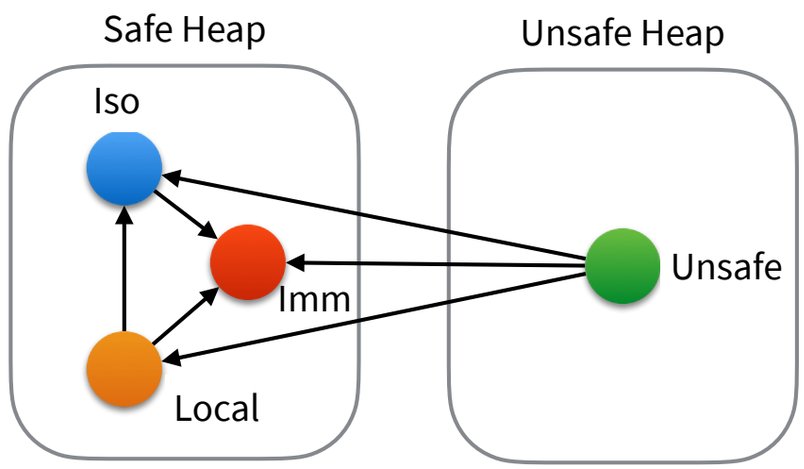
~~Initial code~~ Isolate list

```
ls = iso object List {…}
ls.append(imm object 0 {…} )
```

```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```



Dalarna Programming Model

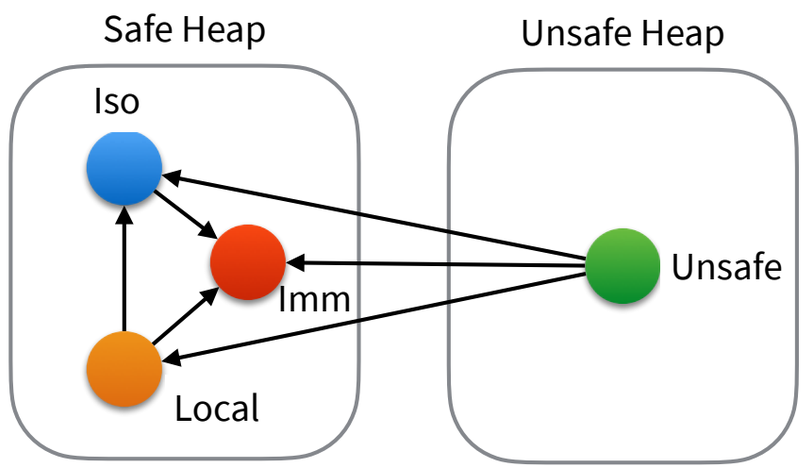
~~Initial code~~ Isolate list

```
ls = iso object List {…}
ls.append(imm object 0 {…} )
```

```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```



Dalarna Programming Model

~~Initial code~~ Isolate list

```
ls = iso object List {…}
ls.append(imm object 0 {…} )
```

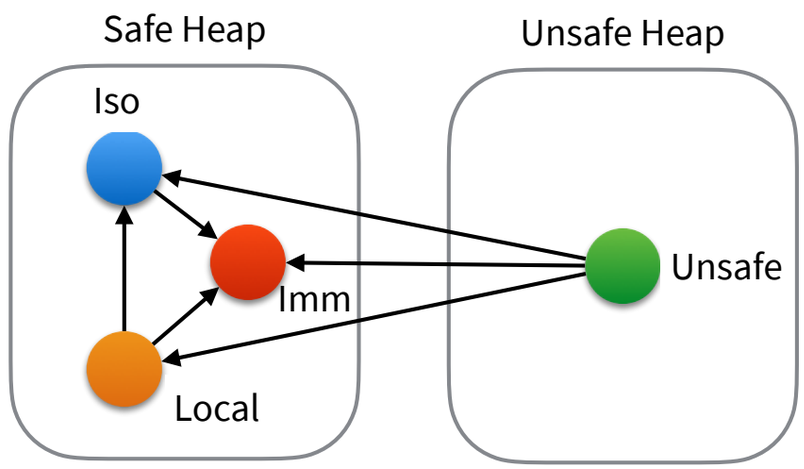
```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Zero copying

TypeScript

Default object considered with **unsafe capability**

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```



Dalarna Programming Model

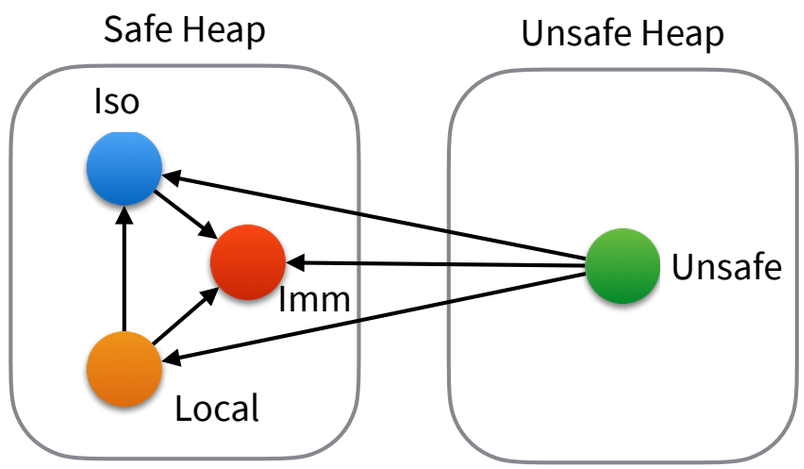
~~Initial code~~ Isolate list

```
ls = iso object List {…}
ls.append(imm object 0 {…} )
```

```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```



Dalarna Programming Model

~~Initial code~~ Isolate list

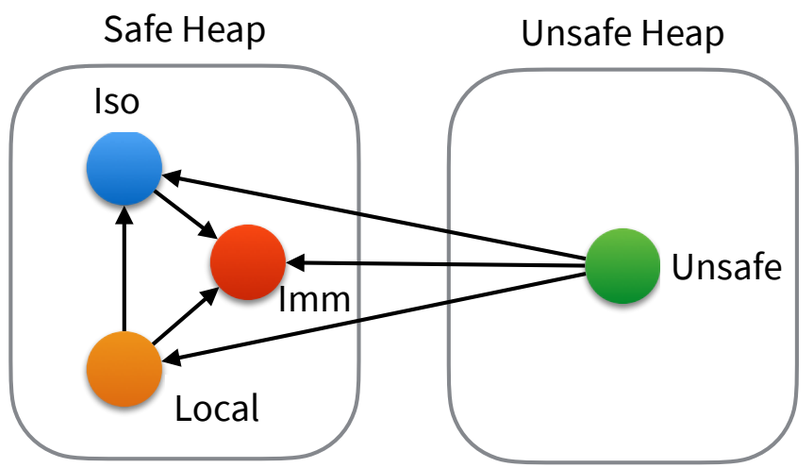
```
ls = iso object List {…}
ls.append(imm object 0 {…} )
```

Default object considered with **unsafe capability**

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```

```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...  -- mutate object
  }
}

ls.map(mapper)
```

Immutable objects cannot be mutated



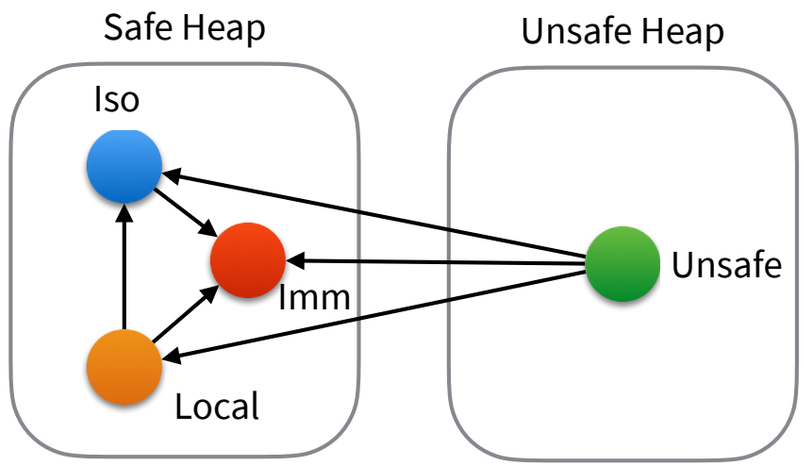
Dalarna Programming Model

~~Initial code~~ Local list

```
ls = local object List {…}
ls.append(local object 0 {…} )

mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
≡
ls = unsafe object List {…}
```



Dalarna Programming Model

~~Initial code~~ Local list

```
ls = local object List {…}
ls.append(local object 0 {…} )
```

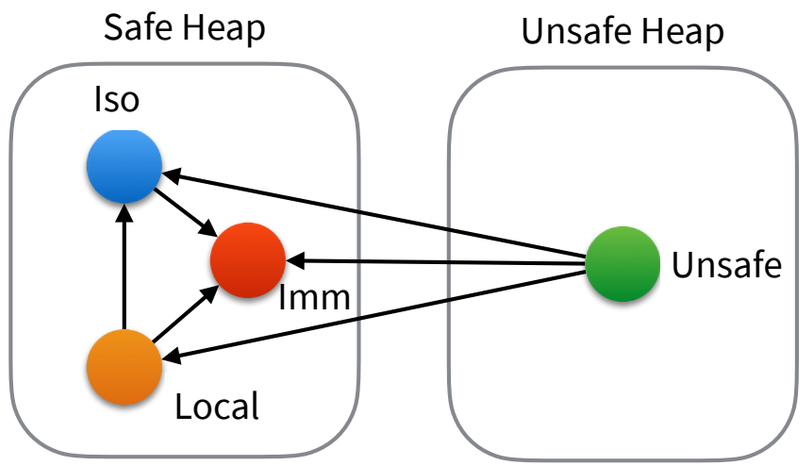
```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
```

≡

```
ls = unsafe object List {…}
```



Dalarna Programming Model

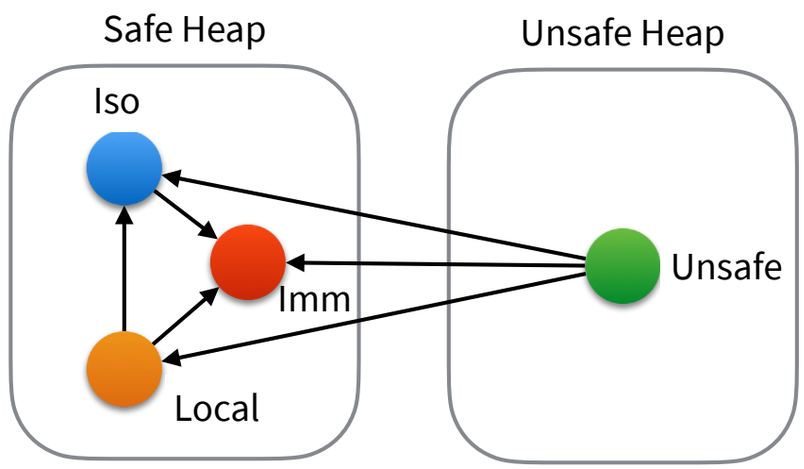
~~Initial code~~ Local list

```
ls = local object List {…}
ls.append(local object 0 {…} )
```

Default object considered with **unsafe capability**

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```

```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```



Dalarna Programming Model

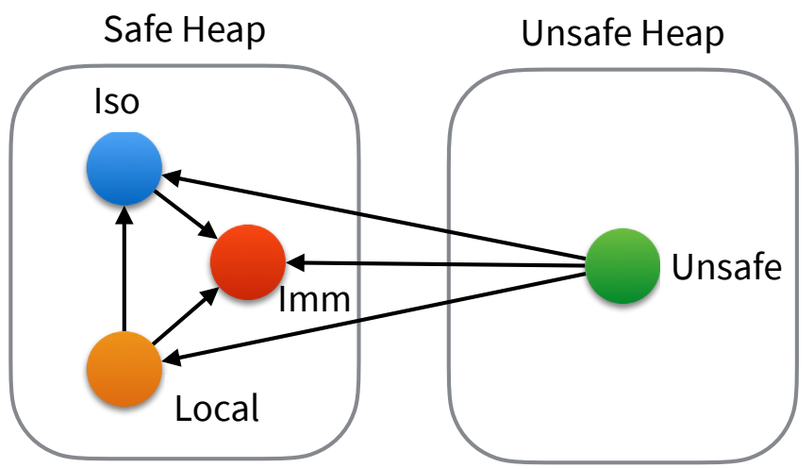
~~Initial code~~ Local list

```
ls = local object List {…}
ls.append(local object 0 {…} )
```

Default object considered with **unsafe capability**

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```

```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```



Dalarna Programming Model

~~Initial code~~ Local list

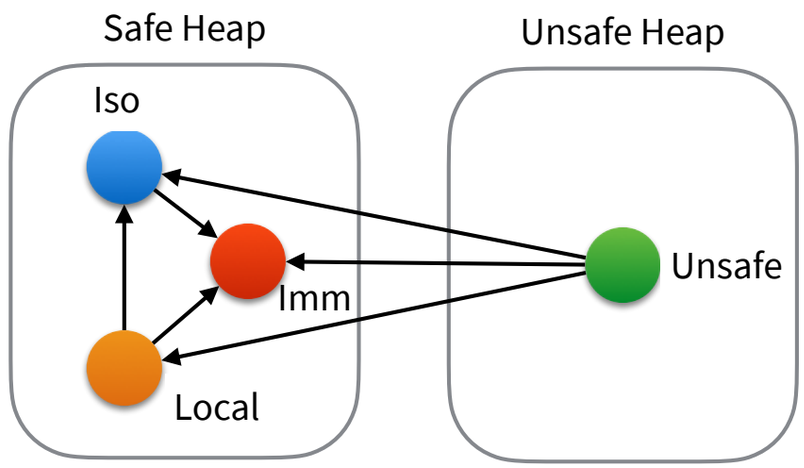
```
ls = local object List {…}
ls.append(local object 0 {…} )
```

Default object considered with **unsafe capability**

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```

```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- o
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Local objects cannot move across threads



Dalarna Programming Model

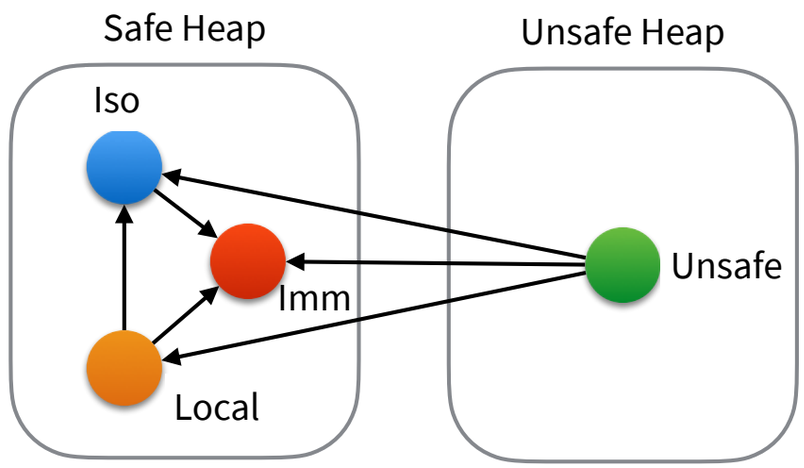
~~Initial code~~ Local list

```
ls = local object List {…}
ls.append(local object 0 {…} )
```

```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <-          copy(o)
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```

Default object considered with **unsafe capability**

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```

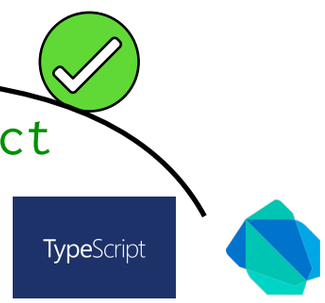


Dalarna Programming Model

~~Initial code~~ Local list

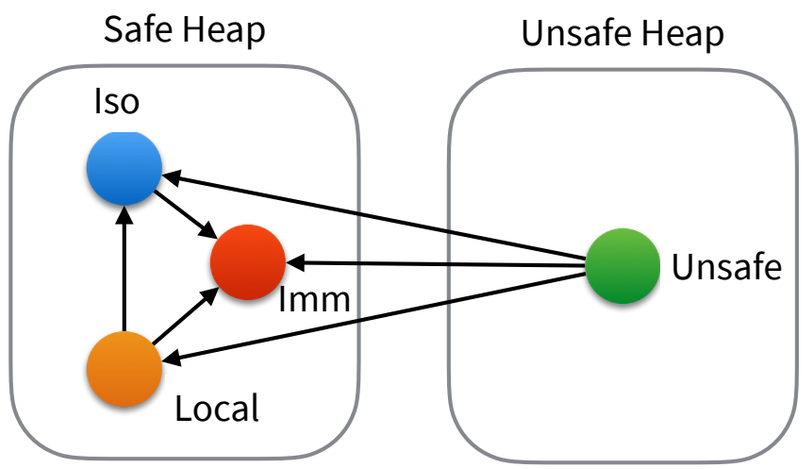
```
ls = local object List {…}
ls.append(local object 0 {…} )
```

```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <-          copy(o)
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```



Default object considered with **unsafe capability**

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```

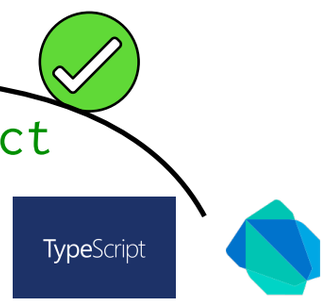


Dalarna Programming Model

~~Initial code~~ Local list

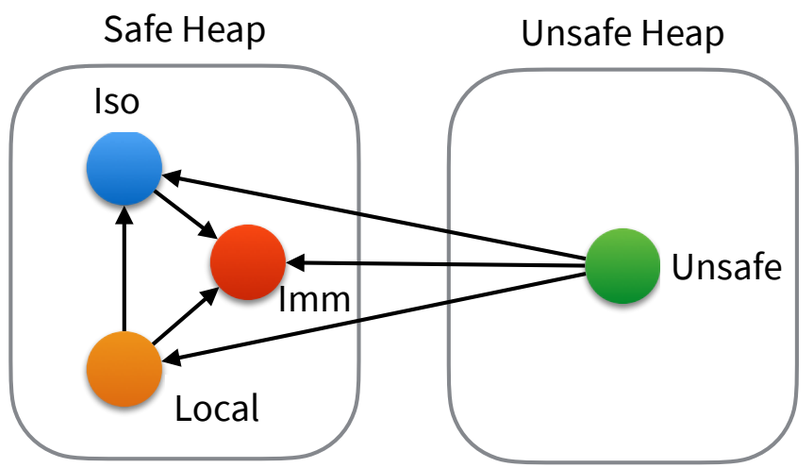
```
ls = local object List {…}
ls.append(local object 0 {…} )
```

```
mapper = object Mapper(...) {
  method map(o) {
    let ch = spawn(ch) {
      oo <- ch
      ...     -- mutate object
    }
    ch <- unsafe copy(o)
    o.f = ...   -- mutate object
  }
}

ls.map(mapper)
```



Default object considered with **unsafe capability**

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```



Dalarna Programming Model

~~Initial code~~ Unsafe Local list

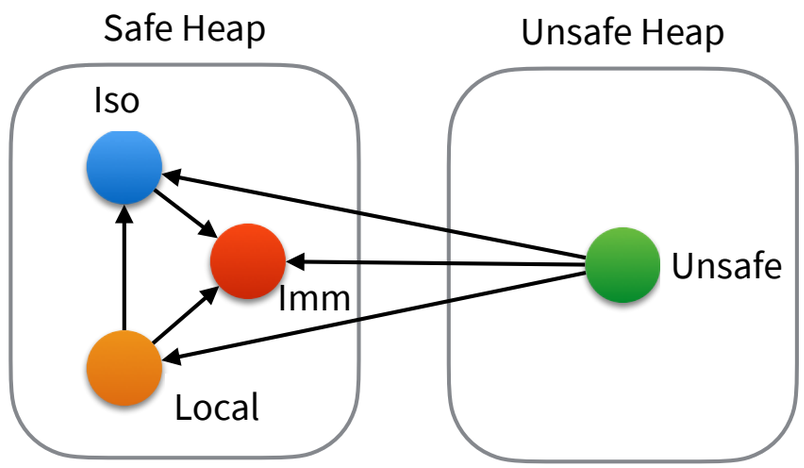
```
ls = unsafe object List {…}
ls.append(local object 0 {…} )

let ch = spawn(ch) {
    oo <- ch
    ...     -- mutate object
}

ch <- ls
```

Default object considered with **unsafe capability**

```
ls = object List {…}
```

≡

```
ls = unsafe object List {…}
```



Dalarna Programming Model

~~Initial code~~ Unsafe Local list

```
ls = unsafe object List {…}
ls.append(local object 0 {…} )
```

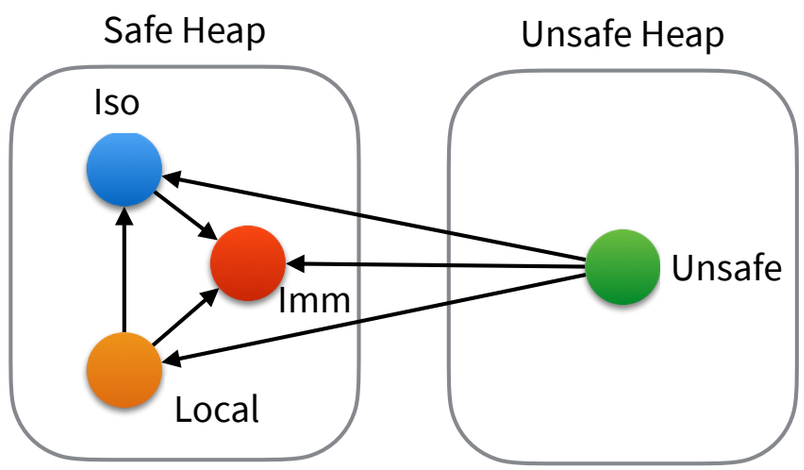
```
let ch = spawn(ch) {
    oo <- ch
    ...     -- mutate object
}

ch <- ls
```

Default object considered with **unsafe capability**

```
ls = object List {…}
```

≡

```
ls = unsafe object List {…}
```



Dalarna Programming Model

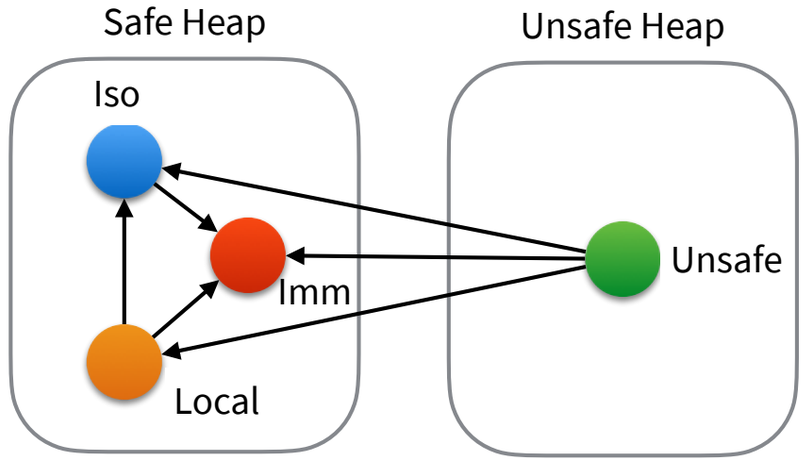
~~Initial code~~ Local list

```
ls = unsafe object List {…}
ls.append(local object 0 {…} )
```

Default object considered with **unsafe capability**

```
let ch = spawn(ch) {
    oo <- ch
    ...     -- mutate object
}

ch <- ls
```

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```



Dalarna Programming Model

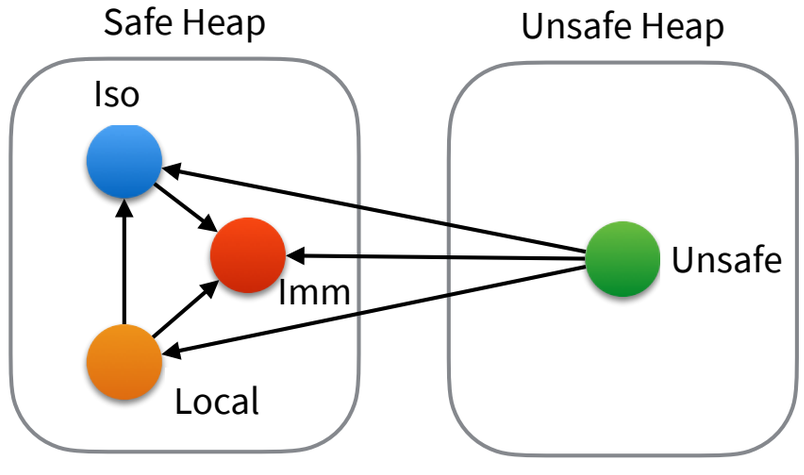
~~Initial code~~ Local list

```
ls = unsafe object List {…}
ls.append(local object 0 {…} )
```

Default object considered with **unsafe capability**

```
let ch = spawn(ch) {
    oo <- ch
    ...     -- mutate object
}

ch <- ls
```

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```



Dalarna Programming Model

~~Initial code~~ Local list

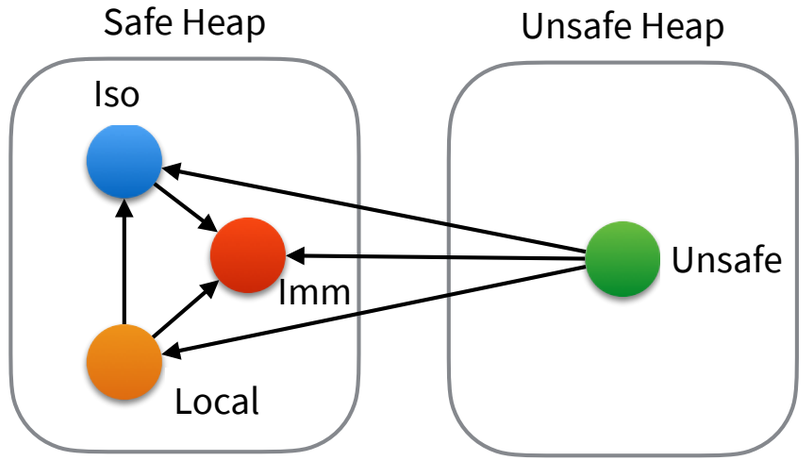
```
ls = unsafe object List {…}
ls.append(local object 0 {…} )
```

Default object considered with **unsafe capability**

```
let ch = spawn(ch) {
    oo <- ch
    ...     -- mutate object
}

ch <- ls
```

Local objects cannot move across threads

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```



Dalarna Programming Model

~~Initial code~~ Local list

```
ls = unsafe object List {…}
ls.append(local object 0 {…} )
```

Default object considered with **unsafe capability**

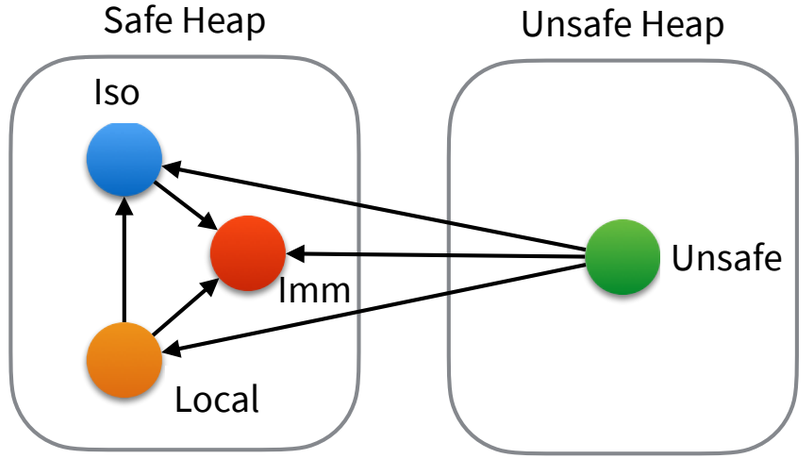
```
let ch = spawn(ch) {
    oo <- ch
    ...     -- mutate object
}

ch <- ls
```

Local objects cannot move across threads

```
ls = object List {…}
          ≡
ls = unsafe object List {…}
```

Main Point:

Any code that can lead
to a data race is a runtime error



Dalarna Dynamic Semantics

$$
\begin{array}{rcll}
\textit{Configuration} & \textit{Cfg} & ::= & H, \overline{T} \\
\textit{Thread} & T & ::= & t \mid \textit{Err} \\
\textit{Thread Err} & \textit{Err} & ::= & \mathrm{Err}_N \mid \mathrm{Err}_A \mid \mathrm{Err}_P \mid \mathrm{Err}_C
\end{array}
$$

Dalarna Dynamic Semantics

$$
\begin{array}{rrcl}
\textit{Configuration} & \textit{Cfg} & ::= & H, \overline{T} \\
\textit{Thread} & T & ::= & t \mid \textit{Err} \\
\textit{Thread Err} & \textit{Err} & ::= & \mathrm{Err}_N \mid \mathrm{Err}_A \mid \mathrm{Err}_P \mid \mathrm{Err}_C
\end{array}
$$

(R-Let)
$$\frac{x \notin dom(H)}{H; \mathbf{let}\ x\ =\ v\ \mathbf{in}\ t \rightsquigarrow H, x \mapsto v; t}$$

(R-Var)
$$\frac{H(x) = v \quad \neg\mathrm{isIso}(H, v)}{H; E[x] \rightsquigarrow H; E[v]}$$

(R-Consume)
$$\frac{H(x) = v \quad H' = H[x \mapsto \top]}{H; E[\mathbf{consume}\ x] \rightsquigarrow H'; E[v]}$$

(R-Field)
$$\frac{H(H(x)) = \_\ \mathbf{obj}\ \{\_\ f = v\ M\} \quad \neg\mathrm{isIso}(H, v)}{H; E[x.f] \rightsquigarrow H; E[v]}$$

(R-Update)
$$\frac{\begin{array}{c}H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \\ \neg\mathrm{isImm}(H, \iota) \quad \mathrm{OkRef}(H, K, v) \\ H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]\end{array}}{H; E[x.f = v] \rightsquigarrow H'; E[v']}$$

(R-CastLoc)
$$\frac{H(\iota) = K\ \mathbf{obj}\ \{\_\_\}}{H; E[(K)\ \iota] \rightsquigarrow H; E[\iota]}$$

(R-New)
$$\frac{\begin{array}{c}\forall f = v \in \overline{f = v}.\ \mathrm{OkRef}(H, K, v) \\ \iota\ fresh \quad H' = H, \iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ \overline{M}\}\end{array}}{H; E[K\ \mathbf{obj}\ \{\overline{f = v}\ \overline{M}\}] \rightsquigarrow H'; E[\iota]}$$

(R-Copy)
$$\frac{\begin{array}{c}\mathbf{iso} \notin K \\ \mathrm{OkDup}(H, K, H(x)) = (H', \iota)\end{array}}{H; E[K\ \mathbf{copy}\ x] \rightsquigarrow H'; E[\iota]}$$

(R-Call)
$$\frac{\begin{array}{c}x', y'\ fresh \quad H(x) = \iota \\ H(\iota) = \_\ \mathbf{obj}\ \{\_\ \overline{M}\ \mathbf{method}\ m(y)\ \{t\}\}\end{array}}{H; E[x.m(v)] \rightsquigarrow H, x' \mapsto \iota, y' \mapsto v; E[t[\mathbf{self} = x'][y = y']]}$$

(R-Spawn)
$$\frac{\begin{array}{c}\iota, i\ fresh \quad x \notin dom(H) \\ H' = H, x \mapsto \iota, \iota \mapsto \mathbf{chan}\ \{i, \varnothing\}\end{array}}{H; E[\mathbf{spawn}\ (x)\ \{t\}] \rightsquigarrow H'; E[\iota]\ t}$$

(R-Recv)
$$\frac{\begin{array}{c}H(\iota) = \mathbf{chan}\ \{i, \iota'\} \\ H' = H[\iota \mapsto \mathbf{chan}\ \{i, \varnothing\}]\end{array}}{H; E[\leftarrow \iota] \rightsquigarrow H'; E[\iota']}$$

(R-SendBlock)
$$\frac{\begin{array}{c}H(\iota) = \mathbf{chan}\ \{\_, \varnothing\} \quad \mathrm{OkSend}(H, v) \\ i\ fresh \quad H' = H[\iota \mapsto \mathbf{chan}\ \{i, v\}]\end{array}}{H; E[\iota \leftarrow v] \rightsquigarrow H'; E[\blacksquare_i\ \iota]}$$

(R-SendUnblock)
$$\frac{\begin{array}{c}H(\iota) = \mathbf{chan}\ \{i', v\} \\ v = \varnothing \vee i \neq i'\end{array}}{H; E[\blacksquare_i\ \iota] \rightsquigarrow H; E[\iota]}$$

(RefCheck)
$$\frac{\begin{array}{c}H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \\ \mathrm{OkField}(K, K')\end{array}}{\mathrm{OkRef}(H, K, \iota)}$$

(Helper-OkSend)
$$\frac{\begin{array}{c}H(\iota) = K\ \mathbf{obj}\ \{\_\_\} \\ \neg\exists \iota' \in \mathrm{ROG}(H, \iota)\ .\ \mathrm{isLocal}(H, \iota')\end{array}}{\mathrm{OkSend}(H, \iota)}$$

(Helper-OkField)
$$\frac{\begin{array}{c}K_3 = \mathrm{Max}(K_1) \\ \forall\, K_4 \in K_2\ .\ K_3 \leq K_4\end{array}}{\mathrm{OkField}(K_1, K_2)}$$

Dalarna Dynamic Semantics

$$
\begin{array}{rrcl}
\textit{Configuration} & \textit{Cfg} & ::= & H, \overline{T} \\
\textit{Thread} & T & ::= & t \mid \textit{Err} \\
\textit{Thread Err} & \textit{Err} & ::= & \mathrm{Err}_N \mid \mathrm{Err}_A \mid \mathrm{Err}_P \mid \mathrm{Err}_C
\end{array}
$$

(R-Let)
$$\frac{x \notin dom(H)}{H; \mathbf{let}\ x\ =\ v\ \mathbf{in}\ t \rightsquigarrow H, x \mapsto v; t}$$

(R-Var)
$$\frac{H(x) = v \quad \neg\mathsf{isIso}(H, v)}{H; E[x] \rightsquigarrow H; E[v]}$$

(R-Consume)
$$\frac{H(x) = v \quad H' = H[x \mapsto \top]}{H; E[\mathbf{consume}\ x] \rightsquigarrow H'; E[v]}$$

(R-Field)
$$\frac{H(H(x)) = \_\ \mathbf{obj}\ \{\_\ f = v\ \overline{M}\} \quad \neg\mathsf{isIso}(H, v)}{H; E[x.f] \rightsquigarrow H; E[v]}$$

(R-Update)
$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}$$

(R-CastLoc)
$$\frac{H(\iota) = K\ \mathbf{obj}\ \{\_\_\}}{H; E[(K)\ \iota] \rightsquigarrow H; E[\iota]}$$

(R-New)
$$\frac{\forall f = v \in \overline{f = v}.\ \mathsf{OkRef}(H, K, v) \quad \iota\ \textit{fresh} \quad H' = H, \iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ \overline{M}\}}{H; E[K\ \mathbf{obj}\ \{\overline{f = v}\ \overline{M}\}] \rightsquigarrow H'; E[\iota]}$$

(R-Copy)
$$\frac{\mathbf{iso} \notin K \quad \mathsf{OkDup}(H, K, H(x)) = (H', \iota)}{H; E[K\ \mathbf{copy}\ x] \rightsquigarrow H'; E[\iota]}$$

(R-Call)
$$\frac{x', y'\ \textit{fresh} \quad H(x) = \iota \quad H(\iota) = \_\ \mathbf{obj}\ \{\_\ \overline{M}\ \mathbf{method}\ m(y)\ \{t\}\}}{H; E[x.m(v)] \rightsquigarrow H, x' \mapsto \iota, y' \mapsto v; E[t[\mathbf{self} = x'][y = y']]}$$

(R-Spawn)
$$\frac{\iota, i\ \textit{fresh} \quad x \notin dom(H) \quad H' = H, x \mapsto \iota, \iota \mapsto \mathbf{chan}\ \{i, \varnothing\}}{H; E[\mathbf{spawn}\ (x)\ \{t\}] \rightsquigarrow H'; E[\iota]\ t}$$

(R-Recv)
$$\frac{H(\iota) = \mathbf{chan}\ \{i, \iota'\} \quad H' = H[\iota \mapsto \mathbf{chan}\ \{i, \varnothing\}]}{H; E[\leftarrow \iota] \rightsquigarrow H'; E[\iota']}$$

(R-SendBlock)
$$\frac{H(\iota) = \mathbf{chan}\ \{\_, \varnothing\} \quad \mathsf{OkSend}(H, v) \quad i\ \textit{fresh} \quad H' = H[\iota \mapsto \mathbf{chan}\ \{i, v\}]}{H; E[\iota \leftarrow v] \rightsquigarrow H'; E[\blacksquare_i\ \iota]}$$

(R-SendUnblock)
$$\frac{H(\iota) = \mathbf{chan}\ \{i', v\} \quad v = \varnothing \vee i \neq i'}{H; E[\blacksquare_i\ \iota] \rightsquigarrow H; E[\iota]}$$

(RefCheck)
$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}$$

(Helper-OkSend)
$$\frac{H(\iota) = K\ \mathbf{obj}\ \{\_\_\} \quad \neg\exists \iota' \in \mathrm{ROG}(H, \iota)\ .\ \mathsf{isLocal}(H, \iota')}{\mathsf{OkSend}(H, \iota)}$$

(Helper-OkField)
$$\frac{K_3 = \mathsf{Max}(K_1) \quad \forall\, K_4 \in K_2\ .\ K_3 \leq K_4}{\mathsf{OkField}(K_1, K_2)}$$

Dalarna Dynamic Semantics

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']} \text{(R-Update)}$$

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)} \text{(RefCheck)}$$

$$\frac{K_3^{Max}, K_3^{Min} = \mathsf{Max}(K_1), \mathsf{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathsf{Max}(K_2), \mathsf{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)} \text{(Helper-OkField)}$$

Dalarna Dynamic Semantics

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']} \text{(R-Update)}$$

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)} \text{(RefCheck)}$$

$$\frac{K_3^{Max}, K_3^{Min} = \mathsf{Max}(K_1), \mathsf{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathsf{Max}(K_2), \mathsf{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)} \text{(Helper-OkField)}$$

Dalarna Dynamic Semantics

(R-Update)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}$$

(RefCheck)

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}$$

(Helper-OkField)

$$\frac{K_3^{Max}, K_3^{Min} = \mathsf{Max}(K_1), \mathsf{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathsf{Max}(K_2), \mathsf{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)}$$

Dalarna Dynamic Semantics

(R-Update)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}$$

(RefCheck)

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}$$

(Helper-OkField)

$$\frac{K_3^{Max}, K_3^{Min} = \mathsf{Max}(K_1), \mathsf{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathsf{Max}(K_2), \mathsf{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)}$$

Dalarna Dynamic Semantics

(R-Update)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}$$

(RefCheck)

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}$$

(Helper-OkField)

$$\frac{K_3^{Max}, K_3^{Min} = \mathsf{Max}(K_1), \mathsf{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathsf{Max}(K_2), \mathsf{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)}$$

Dalarna Dynamic Semantics

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \qquad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \qquad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}\ (\textsc{R-Update})$$

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \qquad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}\ (\textsc{RefCheck})$$

$$\frac{K_3^{Max}, K_3^{Min} = \mathsf{Max}(K_1), \mathsf{Min}(K_1) \qquad K_4^{Max}, K_4^{Min} = \mathsf{Max}(K_2), \mathsf{Min}(K_2) \qquad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)}\ (\textsc{Helper-OkField})$$

Dalarna Dynamic Semantics

(R-Update)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}$$

(RefCheck)

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}$$

(Helper-OkField)

$$\frac{K_3^{Max}, K_3^{Min} = \mathsf{Max}(K_1), \mathsf{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathsf{Max}(K_2), \mathsf{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)}$$

Dalarna Dynamic Semantics

(R-Update)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}$$

(RefCheck)

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}$$

(Helper-OkField)

$$\frac{K_3^{Max}, K_3^{Min} = \mathsf{Max}(K_1), \mathsf{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathsf{Max}(K_2), \mathsf{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)}$$

Dalarna Dynamic Semantics

(R-Update)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}$$

(RefCheck)

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}$$

(Helper-OkField)

$$\frac{K_3^{Max}, K_3^{Min} = \mathsf{Max}(K_1), \mathsf{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathsf{Max}(K_2), \mathsf{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)}$$

Dalarna Dynamic Semantics

(R-Update)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}$$

(RefCheck)

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}$$

(Helper-OkField)

$$\frac{K_3^{Max}, K_3^{Min} = \mathsf{Max}(K_1), \mathsf{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathsf{Max}(K_2), \mathsf{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)}$$

Dalarna Dynamic Semantics

(R-Update)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}$$

(RefCheck)

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}$$

(Helper-OkField)

$$\frac{K_3^{Max}, K_3^{Min} = \mathsf{Max}(K_1), \mathsf{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathsf{Max}(K_2), \mathsf{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)}$$

Dalarna Dynamic Semantics

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \qquad \neg\mathrm{isImm}(H, \iota) \quad \mathrm{OkRef}(H, K, v) \qquad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}\ (\textsc{R-Update})$$

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \qquad \mathrm{OkField}(K, K')}{\mathrm{OkRef}(H, K, \iota)}\ (\textsc{RefCheck})$$

$$\frac{K_3^{Max}, K_3^{Min} = \mathrm{Max}(K_1), \mathrm{Min}(K_1) \qquad K_4^{Max}, K_4^{Min} = \mathrm{Max}(K_2), \mathrm{Min}(K_2) \qquad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathrm{OkField}(K_1, K_2)}\ (\textsc{Helper-OkField})$$

$$\frac{\text{local, iso} = \mathrm{Max}(\text{local iso}),\ \ \mathrm{Min}(\text{local iso}) \qquad \text{local, imm} = \mathrm{Max}(\text{local imm}), \mathrm{Min}(\text{local imm}) \qquad \text{local} \leq \text{local} \quad \text{iso} \leq \text{imm}}{\mathrm{OkField}(\text{local iso}, \text{local imm})}$$

Dalarna Dynamic Semantics

(R-Update)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}$$

(RefCheck)

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}$$

(Helper-OkField)

$$\frac{K_3^{Max}, K_3^{Min} = \mathrm{Max}(K_1), \mathrm{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathrm{Max}(K_2), \mathrm{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)}$$

$$\frac{\text{local, iso} = \mathrm{Max}(\text{local iso}),\ \ \mathrm{Min}(\text{local iso}) \quad \text{local, imm} = \mathrm{Max}(\text{local imm}), \mathrm{Min}(\text{local imm}) \quad \text{local} \leq \text{local} \quad \text{iso} \leq \text{imm}}{\mathsf{OkField}(\text{local iso}, \text{local imm})}$$

Dalarna Dynamic Semantics

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}\ (\textsc{R-Update})$$

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}\ (\textsc{RefCheck})$$

$$\frac{K_3^{Max}, K_3^{Min} = \mathsf{Max}(K_1), \mathsf{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathsf{Max}(K_2), \mathsf{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)}\ (\textsc{Helper-OkField})$$

$$\frac{\text{local, iso} = \mathsf{Max}(\text{local iso}),\ \ \mathsf{Min}(\text{local iso}) \quad \text{local, imm} = \mathsf{Max}(\text{local imm}), \mathsf{Min}(\text{local imm}) \quad \text{local} \leq \text{local} \quad \text{iso} \leq \text{imm}}{\mathsf{OkField}(\text{local iso}, \text{local imm})}$$

Dalarna Dynamic Semantics

(R-Update)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}$$

(RefCheck)

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}$$

(Helper-OkField)

$$\frac{K_3^{Max}, K_3^{Min} = \mathrm{Max}(K_1), \mathrm{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathrm{Max}(K_2), \mathrm{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)}$$

local, iso = Max(local iso), Min(local iso)

local, imm = Max(local imm), Min(local imm)

$$\frac{\text{local} \leq \text{local} \qquad \text{iso} \leq \text{imm}}{\mathsf{OkField}(\text{local iso}, \text{local imm})}$$

Dalarna Dynamic Semantics

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']} \text{(R-Update)}$$

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)} \text{(RefCheck)}$$

$$\frac{K_3^{Max}, K_3^{Min} = \mathrm{Max}(K_1), \mathrm{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathrm{Max}(K_2), \mathrm{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)} \text{(Helper-OkField)}$$

$$\frac{\text{local, iso} = \mathrm{Max}(\text{local iso}),\ \ \mathrm{Min}(\text{local iso}) \quad \text{local, imm} = \mathrm{Max}(\text{local imm}), \mathrm{Min}(\text{local imm}) \quad \text{local} \leq \text{local} \quad \text{iso} \leq \text{imm}}{\mathsf{OkField}(\text{local iso}, \text{local imm})}$$

unsafe ≤ local ≤ iso ≤ imm

Dalarna Dynamic Semantics

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\overline{f = v}\ f = v'\ \overline{M}\} \quad \neg\mathsf{isImm}(H, \iota) \quad \mathsf{OkRef}(H, K, v) \quad H' = H[\iota \mapsto K\ \mathbf{obj}\ \{\overline{f = v}\ f = v\ \overline{M}\}]}{H; E[x.f = v] \rightsquigarrow H'; E[v']}\ (\textsc{R-Update})$$

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad \mathsf{OkField}(K, K')}{\mathsf{OkRef}(H, K, \iota)}\ (\textsc{RefCheck})$$

$$\frac{K_3^{Max}, K_3^{Min} = \mathrm{Max}(K_1), \mathrm{Min}(K_1) \quad K_4^{Max}, K_4^{Min} = \mathrm{Max}(K_2), \mathrm{Min}(K_2) \quad K_3^{Max} \leq K_4^{Max} \quad K_3^{Min} \leq K_4^{Min}}{\mathsf{OkField}(K_1, K_2)}\ (\textsc{Helper-OkField})$$

local, iso = Max(local iso), Min(local iso)

local, imm = Max(local imm), Min(local imm)

$$\frac{\text{local} \leq \text{local} \qquad \text{iso} \leq \text{imm}}{\mathsf{OkField}(\text{local iso}, \text{local imm})}$$

unsafe ≤ local ≤ iso ≤ imm

Dalarna Dynamic Semantics

$$
\begin{array}{rrcl}
\textit{Configuration} & \textit{Cfg} & ::= & H, \overline{T} \\
\textit{Thread} & T & ::= & t \mid \textit{Err} \\
\textit{Thread Err} & \textit{Err} & ::= & \mathrm{Err}_N \mid \mathrm{Err}_A \mid \mathrm{Err}_P \mid \mathrm{Err}_C
\end{array}
$$

(E-NoSuchField)
$$\frac{H(x.f) = \bot}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-NoSuchMethod)
$$\frac{H(x) = \iota \quad H(\iota) = \_\ \textbf{obj}\ \{\_\ \overline{M}\} \quad m \notin \mathit{names}(\overline{M})}{H; E[x.m(v)] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-NoSuchFieldAssign)
$$\frac{H(x.f) = \bot}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-SendBadTargetOrArgument)
$$\frac{H(\iota) = \_\ \textbf{obj}\ \{\_\_\} \vee H(\iota') = \textbf{chan}\ \{\_\}}{H; E[\iota \leftarrow \iota'], T \rightsquigarrow H; \mathrm{Err}_N}$$

(E-RecvBadTarget)
$$\frac{H(\iota) = \_\ \textbf{obj}\ \{\_\_\}}{H; E[\ \leftarrow \iota] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-CastError)
$$\frac{H(\iota) = K'\ \textbf{obj}\ \{\_\_\} \quad K' \neq K}{H; E[(K)\ \iota] \rightsquigarrow H; \mathrm{Err}_C}$$

(E-AbsentVar)
$$\frac{H(x) = \top}{H; E[x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-Consume)
$$\frac{H(x) = \top}{H; E[\textbf{consume}\ x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentTarget)
$$\frac{H(x) = \top}{H; E[x.m(v)] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentTargetAccess)
$$\frac{H(x) = \top}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentFieldAssign)
$$\frac{H(x) = \top}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentCopyTarget)
$$\frac{H(x) = \top}{H; E[K\ \textbf{copy}\ x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-BadFieldAssign)
$$\frac{H(x) = \iota \quad H(\iota) = K\ \textbf{obj}\ \{\_\_\} \quad \mathrm{isImm}(H, \iota) \vee \neg \mathrm{OkRef}(H, K, v)}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-BadInstantiation)
$$\frac{\neg \forall v \in \overline{v}.\mathrm{OkRef}(H, K, v)}{H; E[K\ \textbf{obj}\ \{\overline{f = v}\ \overline{M}\}] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-SendingLocal)
$$\frac{H(v) = \textbf{chan}\ \{\_\} \quad \neg \mathrm{OkSend}(H, \iota)}{H; E[v \leftarrow \iota], T \rightsquigarrow H; \mathrm{Err}_P}$$

(E-AliasIso)
$$\frac{H(x) = \iota \quad \mathrm{isIso}(H, \iota)}{H; E[x] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-IsoField)
$$\frac{H(x.f) = \iota \quad \mathrm{isIso}(H, \iota)}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_P}$$

Dalarna Dynamic Semantics

$$
\begin{array}{rrcl}
\textit{Configuration} & \textit{Cfg} & ::= & H, \overline{T} \\
\textit{Thread} & T & ::= & t \mid \textit{Err} \\
\textit{Thread Err} & \textit{Err} & ::= & \mathrm{Err}_N \mid \mathrm{Err}_A \mid \mathrm{Err}_P \mid \mathrm{Err}_C
\end{array}
$$

(E-NoSuchField)

$$\frac{H(x.f) = \bot}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-NoSuchMethod)

$$\frac{H(x) = \iota \quad H(\iota) = \_\ \textbf{obj}\ \{\_\ \overline{M}\} \quad m \notin \mathit{names}(\overline{M})}{H; E[x.m(v)] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-NoSuchFieldAssign)

$$\frac{H(x.f) = \bot}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-SendBadTargetOrArgument)

$$\frac{H(\iota) = \_\ \textbf{obj}\ \{\_\ \_\} \vee H(\iota') = \textbf{chan}\ \{\_\}}{H; E[\iota \leftarrow \iota'], T \rightsquigarrow H; \mathrm{Err}_N}$$

(E-RecvBadTarget)

$$\frac{H(\iota) = \_\ \textbf{obj}\ \{\_\ \_\}}{H; E[\leftarrow \iota] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-CastError)

$$\frac{H(\iota) = K'\ \textbf{obj}\ \{\_\ \_\} \quad K' \neq K}{H; E[(K)\ \iota] \rightsquigarrow H; \mathrm{Err}_C}$$

(E-AbsentVar)

$$\frac{H(x) = \top}{H; E[x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-Consume)

$$\frac{H(x) = \top}{H; E[\textbf{consume}\ x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentTarget)

$$\frac{H(x) = \top}{H; E[x.m(v)] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentTargetAccess)

$$\frac{H(x) = \top}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentFieldAssign)

$$\frac{H(x) = \top}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentCopyTarget)

$$\frac{H(x) = \top}{H; E[K\ \textbf{copy}\ x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-BadFieldAssign)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \textbf{obj}\ \{\_\ \_\} \quad \mathrm{isImm}(H, \iota) \vee \neg \mathrm{OkRef}(H, K, v)}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-BadInstantiation)

$$\frac{\neg \forall v \in \overline{v}.\mathrm{OkRef}(H, K, v)}{H; E[K\ \textbf{obj}\ \{\overline{f = v}\ \overline{M}\}] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-SendingLocal)

$$\frac{H(v) = \textbf{chan}\ \{\_\} \quad \neg \mathrm{OkSend}(H, \iota)}{H; E[v \leftarrow \iota], T \rightsquigarrow H; \mathrm{Err}_P}$$

(E-AliasIso)

$$\frac{H(x) = \iota \quad \mathrm{isIso}(H, \iota)}{H; E[x] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-IsoField)

$$\frac{H(x.f) = \iota \quad \mathrm{isIso}(H, \iota)}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_P}$$

Dalarna Dynamic Semantics

$$
\begin{array}{rrcl}
\textit{Configuration} & \textit{Cfg} & ::= & H, \overline{T} \\
\textit{Thread} & T & ::= & t \mid \textit{Err} \\
\textit{Thread Err} & \textit{Err} & ::= & \mathrm{Err}_N \mid \mathrm{Err}_A \mid \mathrm{Err}_P \mid \mathrm{Err}_C
\end{array}
$$

(E-NoSuchField)

$$\frac{H(x.f) = \bot}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-NoSuchMethod)

$$\frac{H(x) = \iota \quad H(\iota) = \_\ \mathbf{obj}\ \{\_\ \overline{M}\} \quad m \notin \mathit{names}(\overline{M})}{H; E[x.m(v)] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-NoSuchFieldAssign)

$$\frac{H(x.f) = \bot}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-SendBadTargetOrArgument)

$$\frac{H(\iota) = \_\ \mathbf{obj}\ \{\_\ \_\} \vee H(\iota') = \mathbf{chan}\ \{\_\}}{H; E[\iota \leftarrow \iota'], T \rightsquigarrow H; \mathrm{Err}_N}$$

(E-RecvBadTarget)

$$\frac{H(\iota) = \_\ \mathbf{obj}\ \{\_\ \_\}}{H; E[\ \leftarrow \iota] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-CastError)

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\ \_\} \quad K' \neq K}{H; E[(K)\ \iota] \rightsquigarrow H; \mathrm{Err}_C}$$

(E-AbsentVar)

$$\frac{H(x) = \top}{H; E[x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-Consume)

$$\frac{H(x) = \top}{H; E[\mathbf{consume}\ x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentTarget)

$$\frac{H(x) = \top}{H; E[x.m(v)] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentTargetAccess)

$$\frac{H(x) = \top}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentFieldAssign)

$$\frac{H(x) = \top}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentCopyTarget)

$$\frac{H(x) = \top}{H; E[K\ \mathbf{copy}\ x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-BadFieldAssign)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\_\ \_\} \quad \mathrm{isImm}(H, \iota) \vee \neg \mathrm{OkRef}(H, K, v)}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-BadInstantiation)

$$\frac{\neg \forall v \in \overline{v}.\mathrm{OkRef}(H, K, v)}{H; E[K\ \mathbf{obj}\ \{\overline{f = v}\ \overline{M}\}] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-SendingLocal)

$$\frac{H(v) = \mathbf{chan}\ \{\_\} \quad \neg \mathrm{OkSend}(H, \iota)}{H; E[v \leftarrow \iota], T \rightsquigarrow H; \mathrm{Err}_P}$$

(E-AliasIso)

$$\frac{H(x) = \iota \quad \mathrm{isIso}(H, \iota)}{H; E[x] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-IsoField)

$$\frac{H(x.f) = \iota \quad \mathrm{isIso}(H, \iota)}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_P}$$

Dalarna Dynamic Semantics

$$
\begin{array}{rrcl}
\textit{Configuration} & \textit{Cfg} & ::= & H, \overline{T} \\
\textit{Thread} & T & ::= & t \mid \textit{Err} \\
\textit{Thread Err} & \textit{Err} & ::= & \mathrm{Err}_N \mid \mathrm{Err}_A \mid \mathrm{Err}_P \mid \mathrm{Err}_C
\end{array}
$$

(E-NoSuchField)

$$\frac{H(x.f) = \bot}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-NoSuchMethod)

$$\frac{H(x) = \iota \quad H(\iota) = \_\ \mathbf{obj}\ \{\_\ \overline{M}\} \quad m \notin \mathit{names}(\overline{M})}{H; E[x.m(v)] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-NoSuchFieldAssign)

$$\frac{H(x.f) = \bot}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-SendBadTargetOrArgument)

$$\frac{H(\iota) = \_\ \mathbf{obj}\ \{\_\_\} \vee H(\iota') = \mathbf{chan}\ \{\_\}}{H; E[\iota \leftarrow \iota'], T \rightsquigarrow H; \mathrm{Err}_N}$$

(E-RecvBadTarget)

$$\frac{H(\iota) = \_\ \mathbf{obj}\ \{\_\_\}}{H; E[\leftarrow \iota] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-CastError)

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad K' \neq K}{H; E[(K)\ \iota] \rightsquigarrow H; \mathrm{Err}_C}$$

(E-AbsentVar)

$$\frac{H(x) = \top}{H; E[x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-Consume)

$$\frac{H(x) = \top}{H; E[\mathbf{consume}\ x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentTarget)

$$\frac{H(x) = \top}{H; E[x.m(v)] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentTargetAccess)

$$\frac{H(x) = \top}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentFieldAssign)

$$\frac{H(x) = \top}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentCopyTarget)

$$\frac{H(x) = \top}{H; E[K\ \mathbf{copy}\ x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-BadFieldAssign)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\_\_\} \quad \mathrm{isImm}(H, \iota) \vee \neg \mathrm{OkRef}(H, K, v)}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-BadInstantiation)

$$\frac{\neg \forall v \in \overline{v}.\mathrm{OkRef}(H, K, v)}{H; E[K\ \mathbf{obj}\ \{\overline{f = v}\ \overline{M}\}] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-SendingLocal)

$$\frac{H(v) = \mathbf{chan}\ \{\_\} \quad \neg \mathrm{OkSend}(H, \iota)}{H; E[v \leftarrow \iota], T \rightsquigarrow H; \mathrm{Err}_P}$$

(E-AliasIso)

$$\frac{H(x) = \iota \quad \mathrm{isIso}(H, \iota)}{H; E[x] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-IsoField)

$$\frac{H(x.f) = \iota \quad \mathrm{isIso}(H, \iota)}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_P}$$

Dalarna Dynamic Semantics

$$
\begin{array}{rrcl}
\textit{Configuration} & \textit{Cfg} & ::= & H, \overline{T} \\
\textit{Thread} & T & ::= & t \mid \textit{Err} \\
\textit{Thread Err} & \textit{Err} & ::= & \mathrm{Err}_N \mid \mathrm{Err}_A \mid \mathrm{Err}_P \mid \mathrm{Err}_C
\end{array}
$$

(E-NoSuchField)

$$\frac{H(x.f) = \bot}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-NoSuchMethod)

$$\frac{H(x) = \iota \quad H(\iota) = \_\ \mathbf{obj}\ \{\_\ \overline{M}\} \quad m \notin \mathit{names}(\overline{M})}{H; E[x.m(v)] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-NoSuchFieldAssign)

$$\frac{H(x.f) = \bot}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-SendBadTargetOrArgument)

$$\frac{H(\iota) = \_\ \mathbf{obj}\ \{\_\_\} \vee H(\iota') = \mathbf{chan}\ \{\_\}}{H; E[\iota \leftarrow \iota'], T \rightsquigarrow H; \mathrm{Err}_N}$$

(E-RecvBadTarget)

$$\frac{H(\iota) = \_\ \mathbf{obj}\ \{\_\_\}}{H; E[\ \leftarrow \iota] \rightsquigarrow H; \mathrm{Err}_N}$$

(E-CastError)

$$\frac{H(\iota) = K'\ \mathbf{obj}\ \{\_\_\} \quad K' \neq K}{H; E[(K)\ \iota] \rightsquigarrow H; \mathrm{Err}_C}$$

(E-AbsentVar)

$$\frac{H(x) = \top}{H; E[x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-Consume)

$$\frac{H(x) = \top}{H; E[\mathbf{consume}\ x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentTarget)

$$\frac{H(x) = \top}{H; E[x.m(v)] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentTargetAccess)

$$\frac{H(x) = \top}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentFieldAssign)

$$\frac{H(x) = \top}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-AbsentCopyTarget)

$$\frac{H(x) = \top}{H; E[K\ \mathbf{copy}\ x] \rightsquigarrow H; \mathrm{Err}_A}$$

(E-BadFieldAssign)

$$\frac{H(x) = \iota \quad H(\iota) = K\ \mathbf{obj}\ \{\_\_\} \quad \mathrm{isImm}(H, \iota) \vee \neg \mathrm{OkRef}(H, K, v)}{H; E[x.f = v] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-BadInstantiation)

$$\frac{\neg \forall v \in \overline{v}.\mathrm{OkRef}(H, K, v)}{H; E[K\ \mathbf{obj}\ \{\overline{f = v}\ \overline{M}\}] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-SendingLocal)

$$\frac{H(v) = \mathbf{chan}\ \{\_\} \quad \neg \mathrm{OkSend}(H, \iota)}{H; E[v \leftarrow \iota], T \rightsquigarrow H; \mathrm{Err}_P}$$

(E-AliasIso)

$$\frac{H(x) = \iota \quad \mathrm{isIso}(H, \iota)}{H; E[x] \rightsquigarrow H; \mathrm{Err}_P}$$

(E-IsoField)

$$\frac{H(x.f) = \iota \quad \mathrm{isIso}(H, \iota)}{H; E[x.f] \rightsquigarrow H; \mathrm{Err}_P}$$

Dalarna Properties

Theorem 4.4 (Dalarna is Data-Race Free Modulo Unsafe Objects)**.** *Any data race in Dalarna directly or indirectly involves an unsafe object. A data race is defined as a read/write or write/write access to an object from different threads without any interleaving synchronisation, which in our case means a transfer of the object from one thread to the other.*

Theorem 4.5 (Preservation)**.** *Given a well-formed configuration* $\Gamma; H \vdash t\ \overline{T}$ *and* $H; t\ \overline{T} \rightsquigarrow H'; \overline{T'}\ \overline{T}$ *then, there exists a* Γ' *s.t.* $\Gamma' \supseteq \Gamma$ *and* $\Gamma'; H' \vdash \overline{T'}\ \overline{T}$

Theorem 4.6 (Progress)**.** *Given a well-formed configuration* $\Gamma; H \vdash \overline{T}$*, then either* $\Gamma; H \vdash \overline{T}$ *is a terminal configuration (see below) or* $H; \overline{T} \rightsquigarrow H'; \overline{T'}$*.*

Dalarna Properties

Theorem 4.4 (Dalarna is Data-Race Free Modulo Unsafe Objects). *Any data race in Dalarna directly or indirectly involves an unsafe object. A data race is defined as a read/write or write/write access to an object from different threads without any interleaving synchronisation, which in our case means a transfer of the object from one thread to the other.*

Theorem 4.5 (Preservation). *Given a well-formed configuration* $\Gamma; H \vdash t\ \overline{T}$ *and* $H; t\ \overline{T} \rightsquigarrow H'; \overline{T'}\ \overline{T}$ *then, there exists a* Γ' *s.t.* $\Gamma' \supseteq \Gamma$ *and* $\Gamma'; H' \vdash \overline{T'}\ \overline{T}$

Theorem 4.6 (Progress). *Given a well-formed configuration* $\Gamma; H \vdash \overline{T}$, *then either* $\Gamma; H \vdash \overline{T}$ *is a terminal configuration (see below) or* $H; \overline{T} \rightsquigarrow H'; \overline{T'}$.

Dalarna Properties

Theorem 4.4 (Dalarna is Data-Race Free Modulo Unsafe Objects). *Any data race in Dalarna directly or indirectly involves an unsafe object. A data race is defined as a read/write or write/write access to an object from different threads without any interleaving synchronisation, which in our case means a transfer of the object from one thread to the other.*

Theorem 4.5 (Preservation). *Given a well-formed configuration* $\Gamma; H \vdash t\ \overline{T}$ *and* $H; t\ \overline{T} \rightsquigarrow H'; \overline{T'}\ \overline{T}$ *then, there exists a* Γ' *s.t.* $\Gamma' \supseteq \Gamma$ *and* $\Gamma'; H' \vdash \overline{T'}\ \overline{T}$

Theorem 4.6 (Progress). *Given a well-formed configuration* $\Gamma; H \vdash \overline{T}$*, then either* $\Gamma; H \vdash \overline{T}$ *is a terminal configuration (see below) or* $H; \overline{T} \rightsquigarrow H'; \overline{T'}$.

Future Work

- Addition of capability-based gradual type system to statically reject ill-capable programs

```
ls = iso object List {…}
ls.append(unsafe object 0 {…} )
```
Statically rejected

- Improve the prototype and benchmark performance

Transient Typechecks Are (Almost) Free. ECOOP 2019

Richard Roberts, Stefan Marr, Michael Homer, James Noble

Conclusions

- Dalarna programming model
  - guarantees data race free programs
  - safe shared memory without deep copying
  - can be embedded in existing languages (Grace)

Image Source

"Dala Horse" Art Prints by JoniandCo | Redbubble

https://dlpng.com/png/1244869