

Rely-guarantee thinking & separation logic

Cliff Jones, U. Newcastle

Viktor Vafeiadis, MPI-SWS

Overview

Separation & Interference

- Resource invariants
- Ownership transfer
- Rely-guarantee conditions
- Stability

Schedule

- Part I. Separation logic (Viktor)
- Part II. Rely-guarantee (Cliff)
- Part III. RGSep (Viktor)
- Concluding remarks

Separation in Hoare logic: Two important rules

Rule of constancy

$$\frac{\{P\} C \{Q\}}{\{P \wedge R\} C \{Q \wedge R\}} \quad \text{fv}(R) \cap \text{mod}(C) = \emptyset$$

Disjoint parallelism rule

$$\frac{\{P_1\} C_1 \{Q_1\} \quad \{P_2\} C_2 \{Q_2\}}{\{P_1 \wedge P_2\} C_1 \parallel C_2 \{Q_1 \wedge Q_2\}} \quad \begin{array}{l} \text{fv}(P_1, C_1, Q_1) \cap \text{mod}(C_2) = \emptyset \\ \text{fv}(P_2, C_2, Q_2) \cap \text{mod}(C_1) = \emptyset \end{array}$$

What about programs with pointers?

Part I. Separation logic

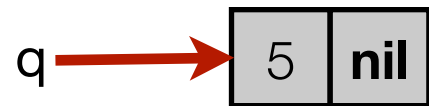
Points-to assertions

SL: convenient syntax for describing the heap



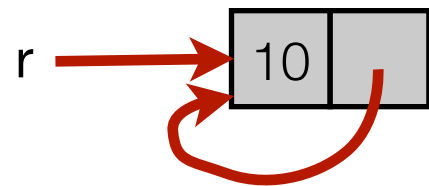
$p \mapsto 5$

$\text{heap}(p) = 5$



$q \mapsto 5, \text{nil}$

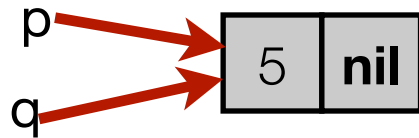
$\text{heap}(q) = 5 \wedge$
 $\text{heap}(q+1) = \text{nil}$



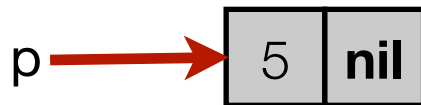
$r \mapsto 10, r$

Conjunction

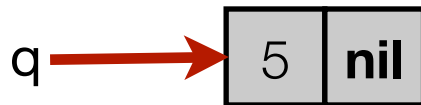
Q: *What does $p \mapsto 5, \text{nil} \wedge q \mapsto 5, \text{nil}$ denote?*



$p \mapsto 5, \text{nil} \wedge p = q$

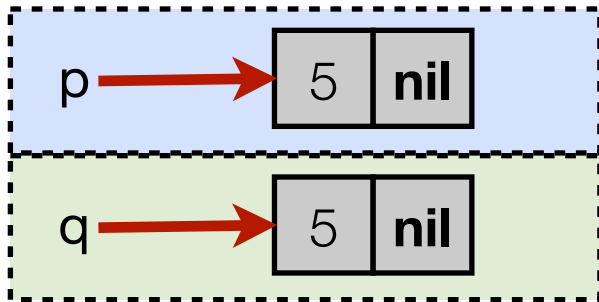


$p \mapsto 5, \text{nil} \wedge q \mapsto 5, \text{nil} \wedge p \neq q$

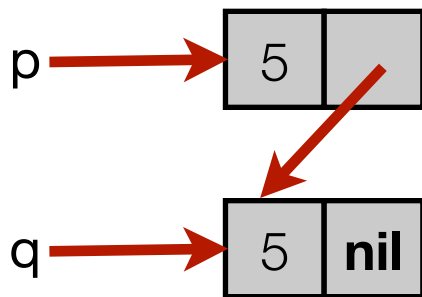


$p \mapsto 5, \text{nil} * q \mapsto 5, \text{nil}$

Separating conjunction

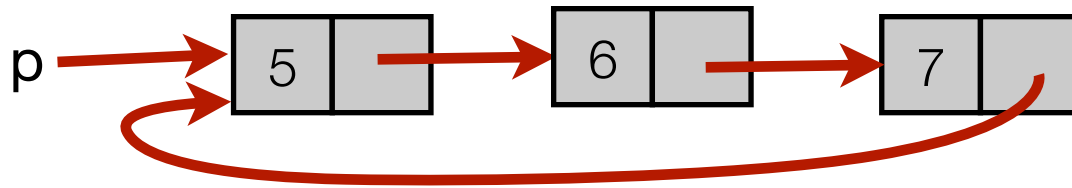


$p \mapsto 5, \text{nil} * q \mapsto 5, \text{nil}$



$p \mapsto 5, q * q \mapsto 5, \text{nil}$

A bigger assertion



$\exists q r. p \mapsto 5, q * q \mapsto 6, r * r \mapsto 7, p$

Classical vs intuitionistic SL

Classical SL

$p \mapsto 5$

$\text{heap}(p) = 5 \wedge \text{dom}(\text{heap}) = \{ p \}$

emp

$\text{dom}(\text{heap}) = \emptyset$

Intuitionistic SL

$p \mapsto 5$

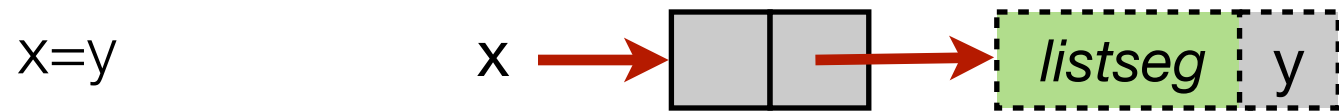
$\text{heap}(p) = 5$

emp

true

$P \iff P * \text{true}$

Inductive definitions: list segments



$$\text{listseg}(x,y) \stackrel{\text{def}}{\iff} x = y \vee \exists v z. x \mapsto v, z * \text{listseg}(z,y)$$

Q: *What does $\text{listseg}(p,p)$ denote?*

Separation logic triples

Just as in Hoare logic...

$$\{ P \} C \{ Q \}$$

... but the precondition must specify all cells the program accesses:

✗ $\{ \text{true} \} [p] := 10 \{ \text{true} \}$

✓ $\{ p \mapsto 5 \} [p] := 10 \{ p \mapsto 10 \}$

✓ $\{ p \mapsto 5 * q \mapsto 6 \} [p] := 10 \{ \text{true} \}$

Frame rule

$$\frac{\{P\} C \{Q\}}{\{P * R\} C \{Q * R\}} \quad \text{fv}(R) \cap \text{mod}(C) = \emptyset$$

Example:

$$\frac{\{p \mapsto 5\} [p] := 10 \{p \mapsto 10\}}{\{p \mapsto 5 * q \mapsto 6\} [p] := 10 \{p \mapsto 10 * q \mapsto 6\}}$$

Disjoint concurrency

$$\frac{\{P_1\} C_1 \{Q_1\} \quad \{P_2\} C_2 \{Q_2\}}{\{P_1 * P_2\} C_1 \parallel C_2 \{Q_1 * Q_2\}} \quad \begin{array}{l} \text{fv}(P_1, C_1, Q_1) \cap \text{mod}(C_2) = \emptyset \\ \text{fv}(P_2, C_2, Q_2) \cap \text{mod}(C_1) = \emptyset \end{array}$$

Well-specified processes 'mind their own business'



Parallel merge sort

```
list(p)
mergesort (p) {
  ...
}
sorted(p)
```

```
{ list(p) } split(p, q) { list(p) * list(q) }
```

```
{ sorted(p) * sorted(q) } merge(p, q) { sorted(p) }
```

Exercise: *Define predicates list(p) and sorted(p).*

Parallel merge sort

list(p)

```
mergesort (p) { local q;
```

```
  ...
```

```
  if ( ... ) {
```

```
    split (p, q);
```

```
    mergesort (p) || mergesort (q) ;
```

```
    merge (p, q);
```

```
  }
```

```
}
```

{ list(p) } split(p, q) { list(p) * list(q) }

{ sorted(p) * sorted(q) } merge(p, q) { sorted(p) }

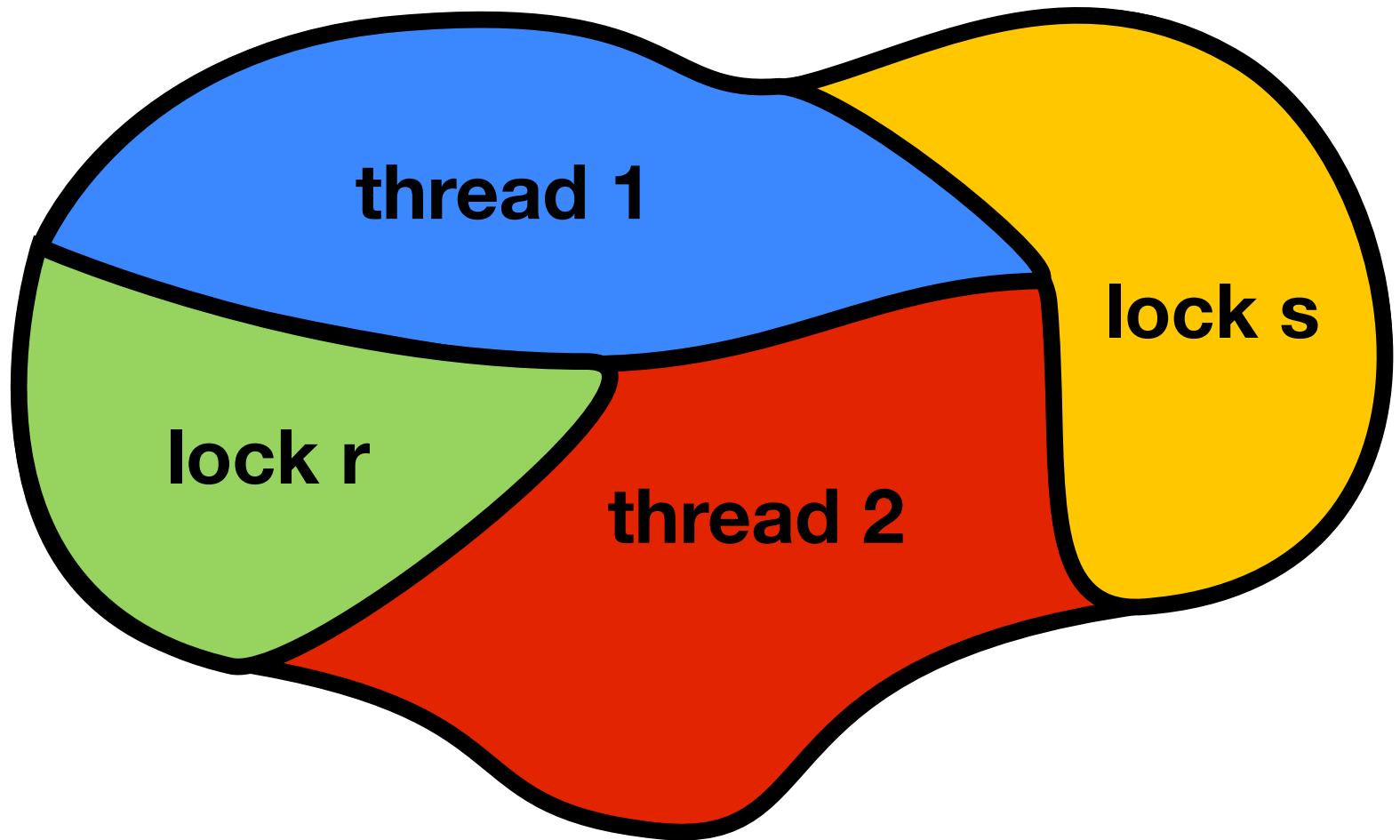
sorted(p)

Parallel merge sort

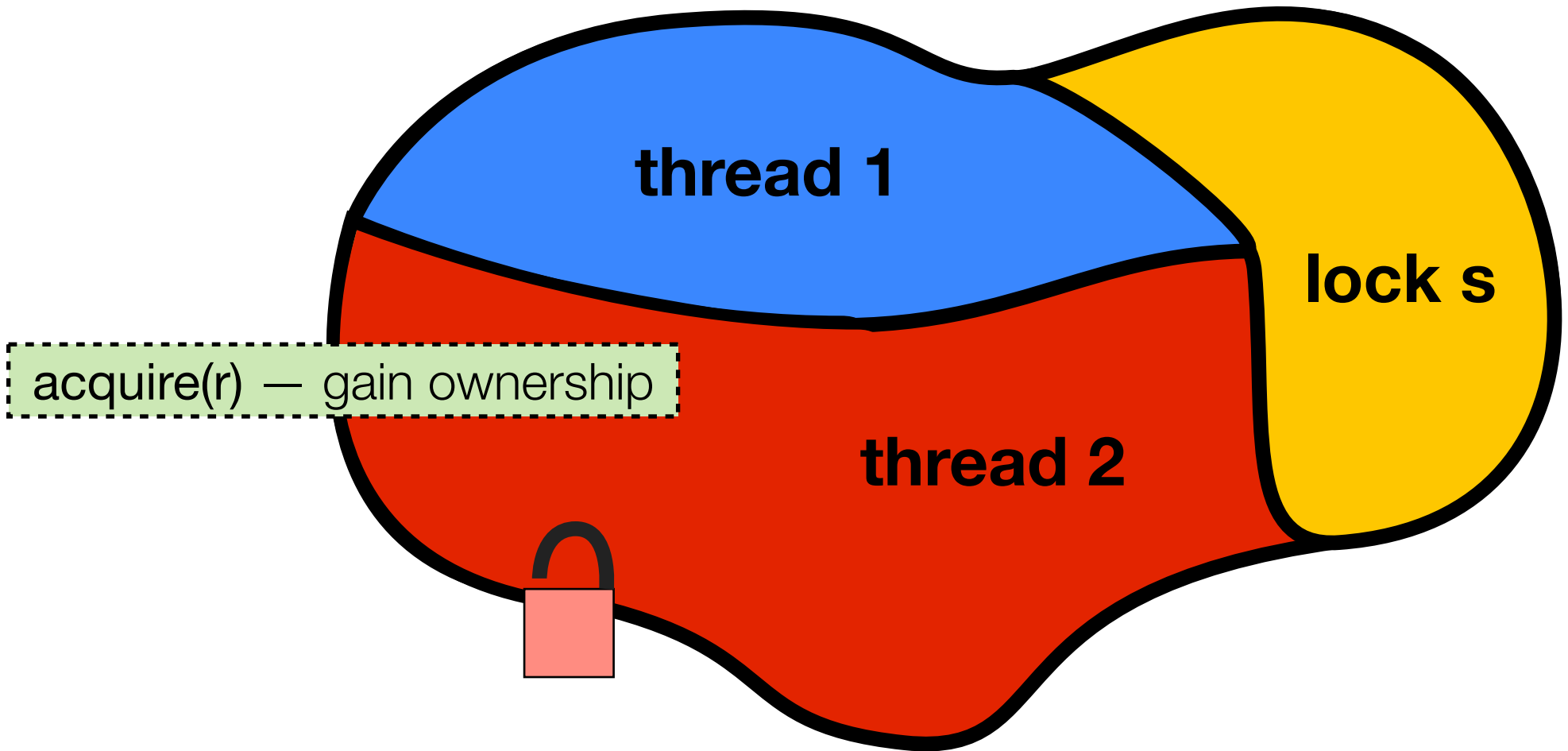
```
list(p)
mergesort (p) { local q; ...
  if ( ... ) {
    list(p)
    split (p, q);
    list(p) * list(q)
    mergesort (p) || mergesort (q) ;
    sorted(p) * sorted(q)
    merge (p, q);
    sorted(p)
  }
}
sorted(p)
```

(Note: In the original image, the split and merge blocks are highlighted in green, and the list and sorted blocks are highlighted in grey.)

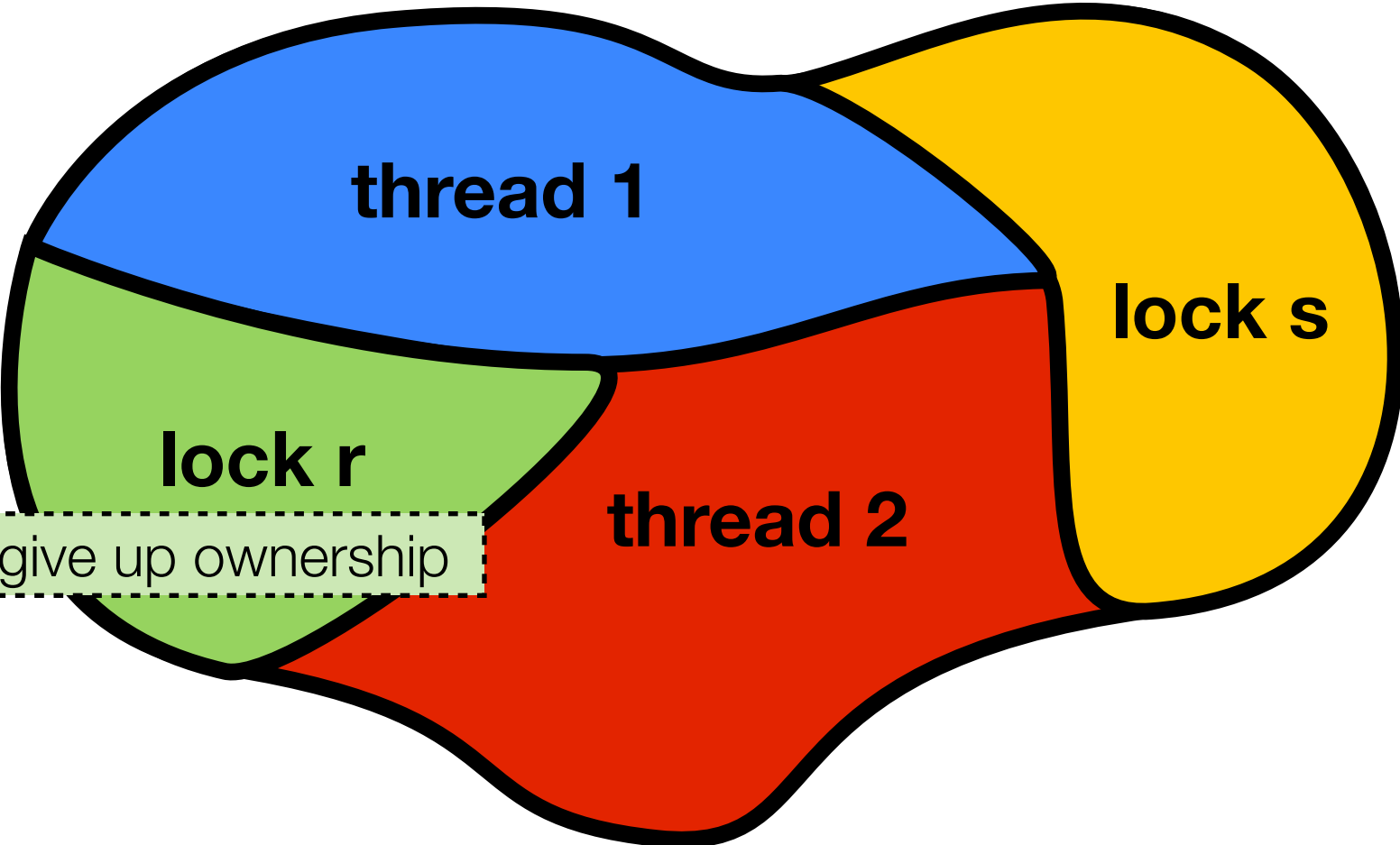
Resource invariants & ownership transfer



Resource invariants & ownership transfer



Resource invariants & ownership transfer



Resource invariants

$$\frac{\Gamma, r: R \vdash \{P\} C \{Q\}}{\Gamma \vdash \{P * R\} \mathbf{resource} \ r \ \mathbf{in} \ C \{Q * R\}}$$

Resource declaration

$$\frac{\Gamma \vdash \{(P * R) \wedge B\} C \{Q * R\}}{\Gamma, r: R \vdash \{P\} \mathbf{with} \ r \ \mathbf{when} \ B \ \mathbf{do} \ C \{Q\}}$$

Resource usage

NB: *Variable side conditions elided.*

Maintaining a data structure invariant

true	p := mkList()	sorted(p)
sorted(p)	insert(p,v)	sorted(p)
sorted(p)	remove(p,v)	sorted(p)

true

p := mkList (); **resource** r **in**

with r **do** insert(p,7)
with r **do** insert(p,10)

||

with r **do** insert(p,5)
with r **do** remove(p,10)

end

sorted(p)

Maintaining a data structure invariant

`true` `p := mkList ();` `sorted(p)` **resource** `r` **in** `true`

Resource Invariant: `sorted(p)`

`true`

with `r` **do** `insert(p,7)`

`true`

with `r` **do** `insert(p,10)`

`true`

end

`sorted(p)`

`true`

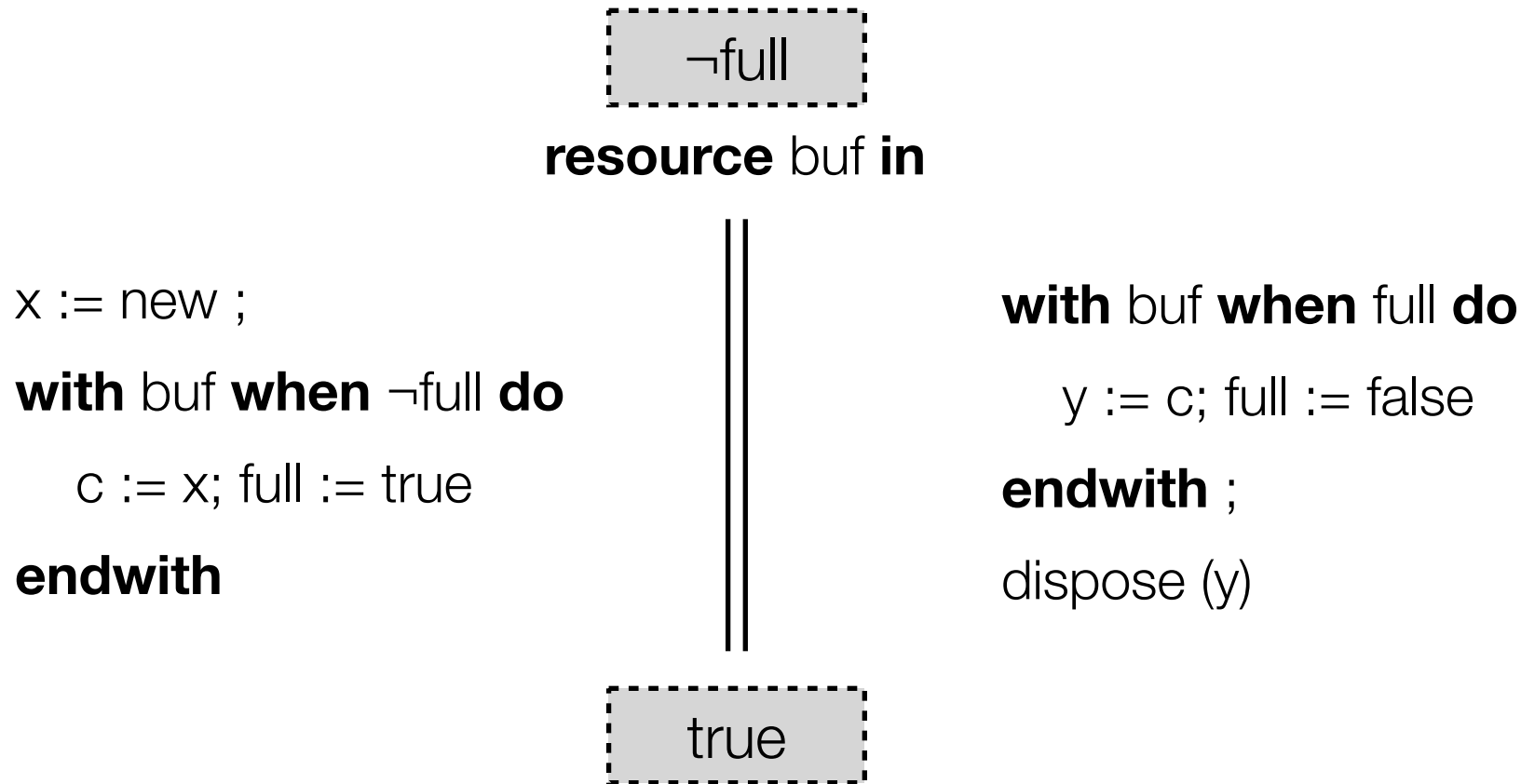
with `r` **do** `insert(p,5)`

`true`

with `r` **do** `remove(p,10)`

`true`

Pointer-transferring buffer



Pointer-transferring buffer

Resource Invariant:

$\neg \text{full} \vee (c \mapsto _ \wedge \text{full})$

true

$x := \text{new};$

$x \mapsto _$

with buf **when** $\neg \text{full}$ **do**

$x \mapsto _ \wedge \neg \text{full}$

$c := x; \text{full} := \text{true}$

$c \mapsto _ \wedge \text{full}$

endwith

true

true

with buf **when** full **do**

$c \mapsto _ \wedge \text{full}$

$y := c; \text{full} := \text{false}$

$y \mapsto _ \wedge \neg \text{full}$

endwith ;

$y \mapsto _$

dispose (y)

true

Tools for separation logic

Inside interactive theorems provers: Coq, HOL4, Isabelle/HOL

- User writes specs & proof (using tactics)

Stand-alone program verifiers: Smallfoot, Verifast, Jstar

- User writes specs & loop invariants

Shape analyses: Space Invader, Thor, Xisa, SLAyer

- User writes specs only (or nothing!)

Mostly just sequential programs

Answers to questions

Q: *What does listseg(p,p) denote?*

A: Either the empty state or a cyclic list.

$\text{list}(p) \stackrel{\text{def}}{\iff} p = \text{nil} \vee \exists v z. x \mapsto v, z * \text{list}(p)$

$\text{sorted}(p, u) \stackrel{\text{def}}{\iff} p = \text{nil} \vee \exists v > u. \exists z. x \mapsto v, z * \text{sorted}(z, v)$

$\text{sorted}(p) \stackrel{\text{def}}{\iff} \text{sorted}(p, -\infty)$