Separation in Hoare logic: Two important rules

Rule of constancy

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

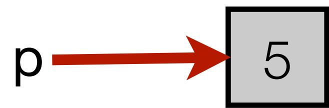
Disjoint parallelism rule

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

What about programs with pointers?

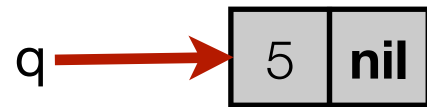
Points-to assertions

SL: convenient syntax for describing the heap



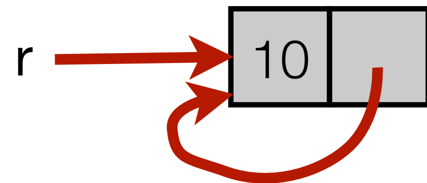
$p \mapsto 5$

heap(p) = 5



$q \mapsto 5, nil$

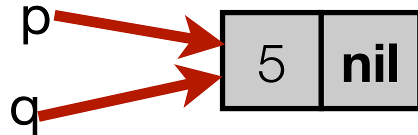
heap(q) = 5 ∧
heap(q+1) = nil



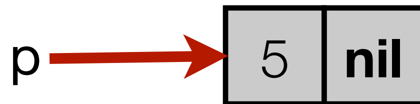
$r \mapsto 10, r$

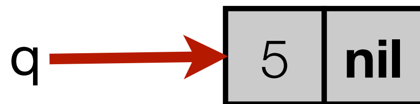
Conjunction

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



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

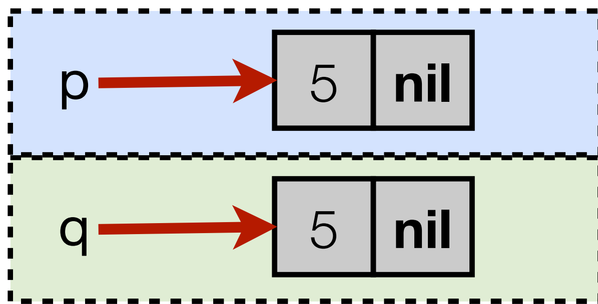




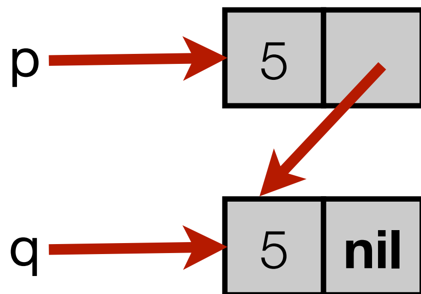
$p \mapsto 5,nil \ \wedge q \mapsto 5,nil \wedge p \neq q$

$p \mapsto 5,nil * q \mapsto 5,nil$

Separating conjunction

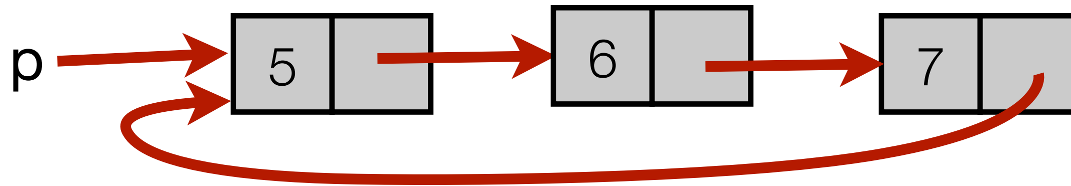


$p \mapsto 5,nil * q \mapsto 5,nil$



$p \mapsto 5,q * q \mapsto 5,nil$

A bigger assertion



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

Classical vs intuitionistic SL

Classical SL

$p \mapsto 5$ $\quad$ $heap(p) = 5 \;\wedge\; dom(heap) = \{ p \}$

emp $\quad$ $dom(heap) = \varnothing$

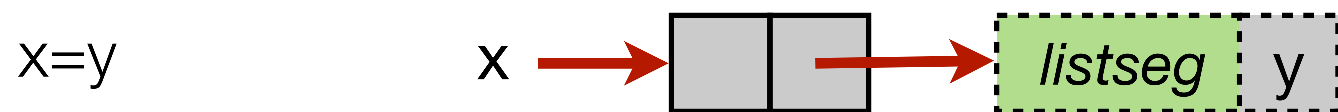
Intuitionistic SL

$p \mapsto 5$ $\quad$ $heap(p) = 5$

emp $\quad$ true

$P \Leftrightarrow P * true$

Inductive definitions: list segments



$$\mathrm{listseg}(x,y) \overset{\text{def}}{\Leftrightarrow} x = y \;\vee\; \exists v\, z.\; x \mapsto v,z * \mathrm{listseg}(z,y)$$

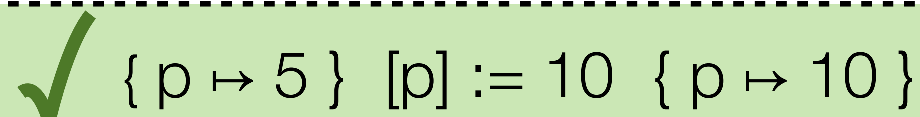
Q: *What does* listseg(p,p) *denote?*

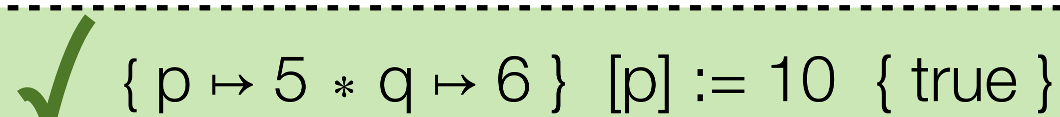
Separation logic triples

Just as in Hoare logic...

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

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





Frame rule

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

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

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

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)

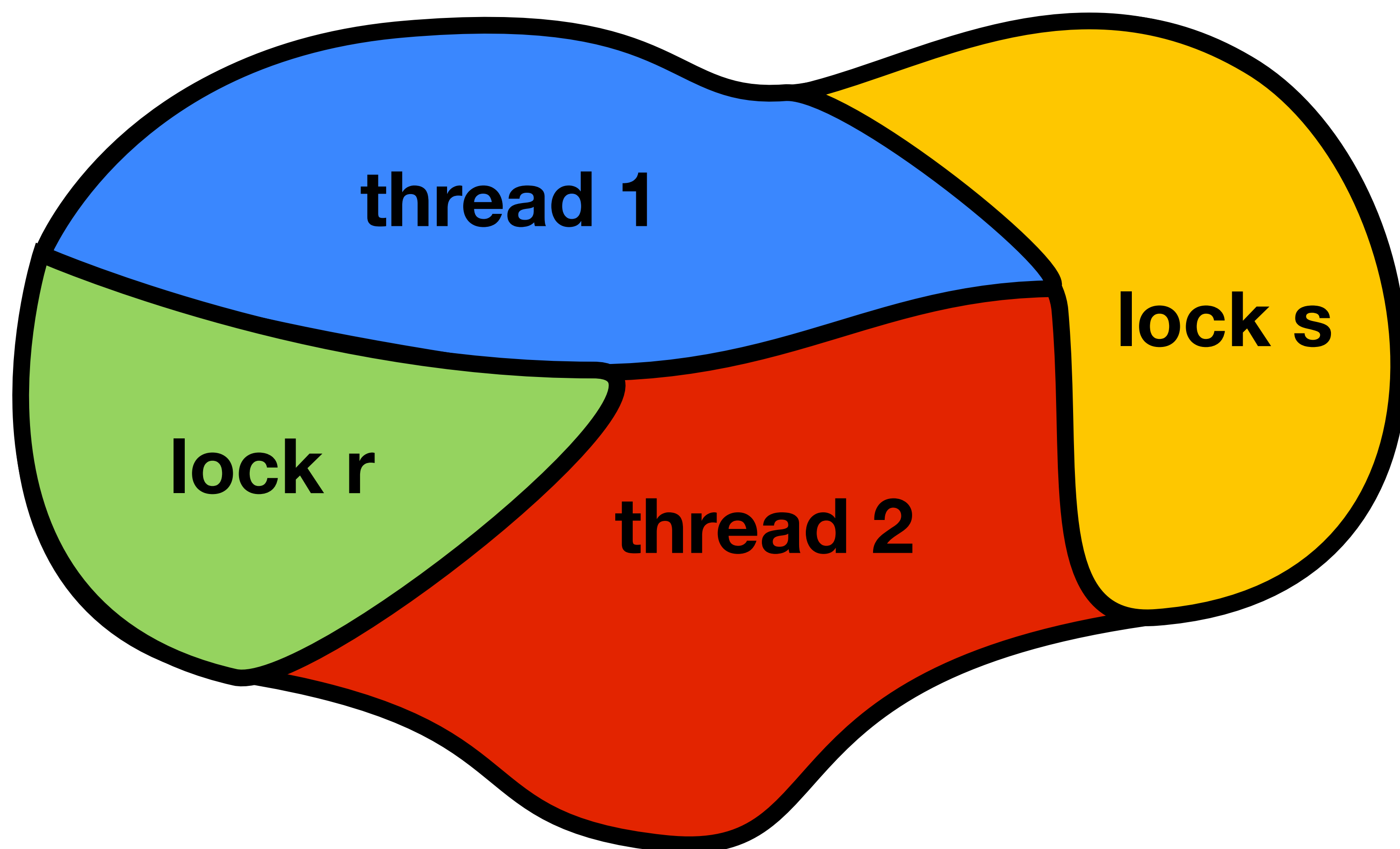
Parallel merge sort

```
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)
```

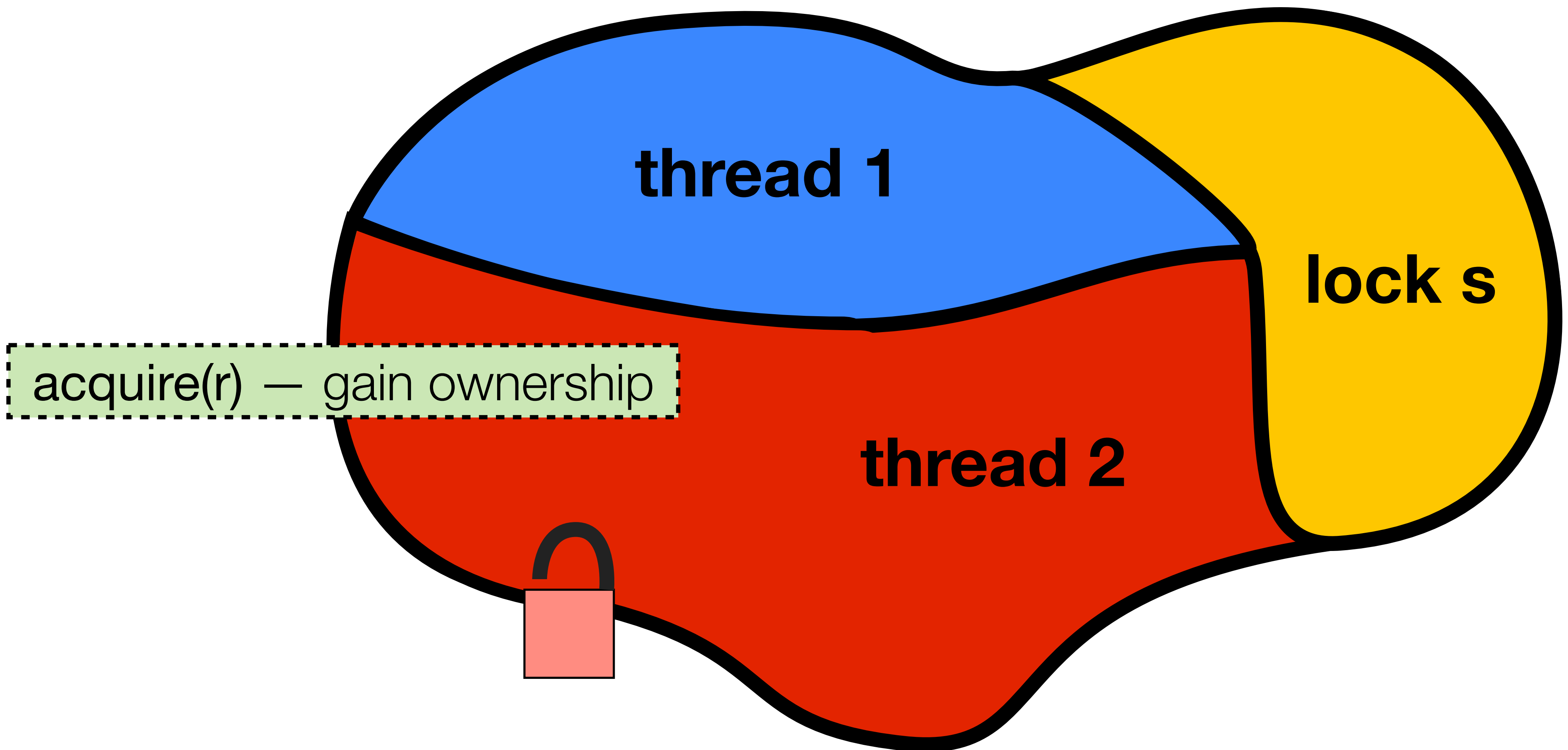
{ list(p) } **split**(p, q) { list(p) ∗ list(q) }

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

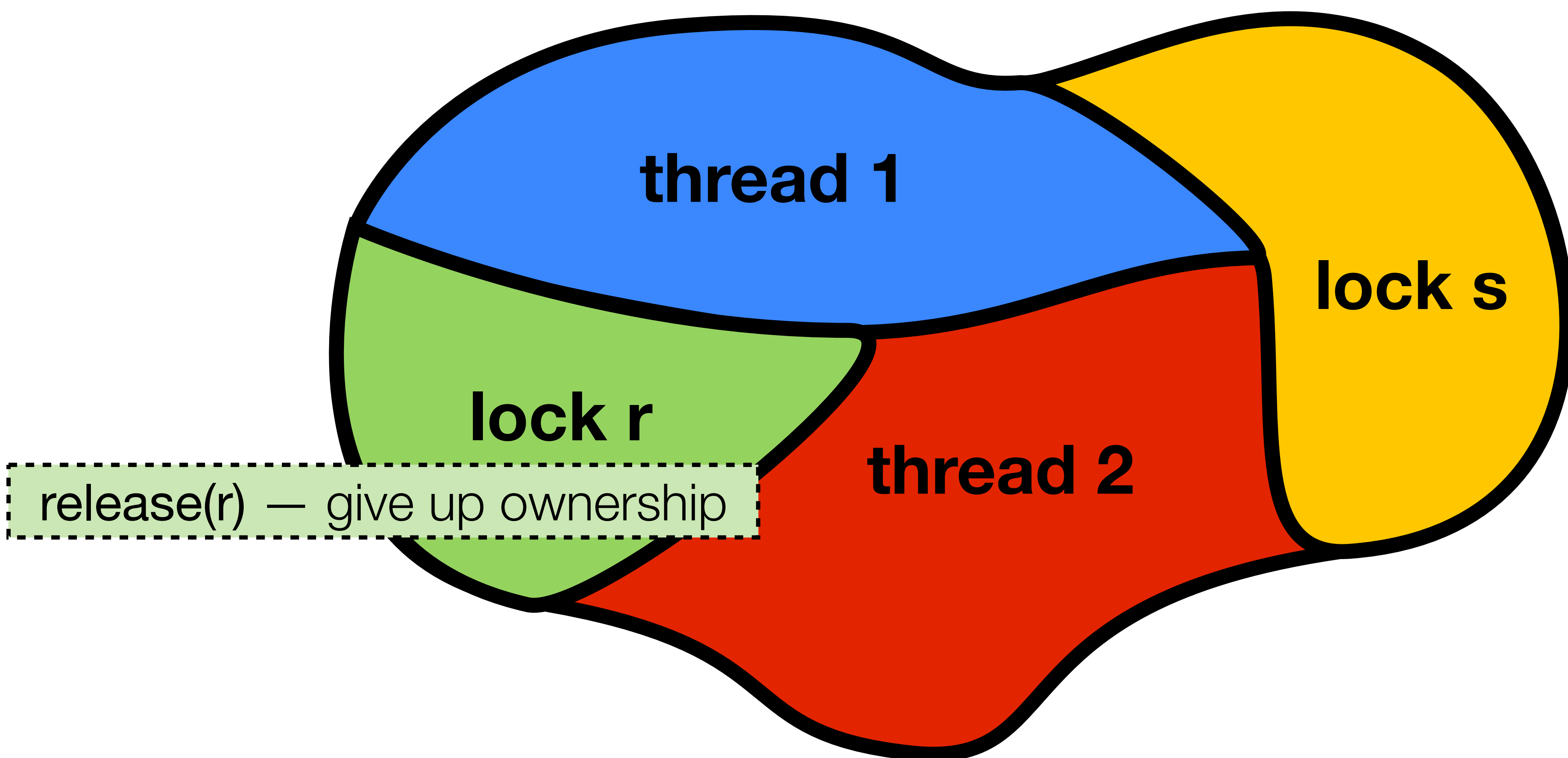
Resource invariants & ownership transfer



Resource invariants & ownership transfer



Resource invariants & ownership transfer



Resource invariants

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

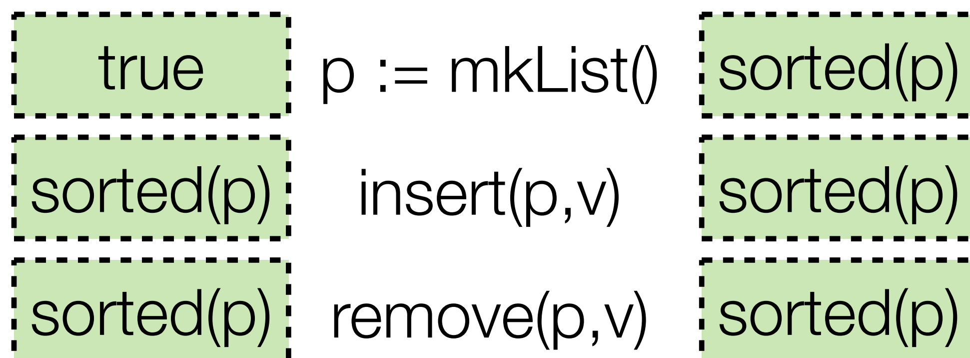
Resource declaration

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

Resource usage

NB: *Variable side conditions elided.*

Maintaining a data structure invariant



true

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

with r **do** insert(p,7) || **with** r **do** insert(p,5)
with r **do** insert(p,10) || **with** r **do** remove(p,10)

end

sorted(p)

Maintaining a data structure invariant

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

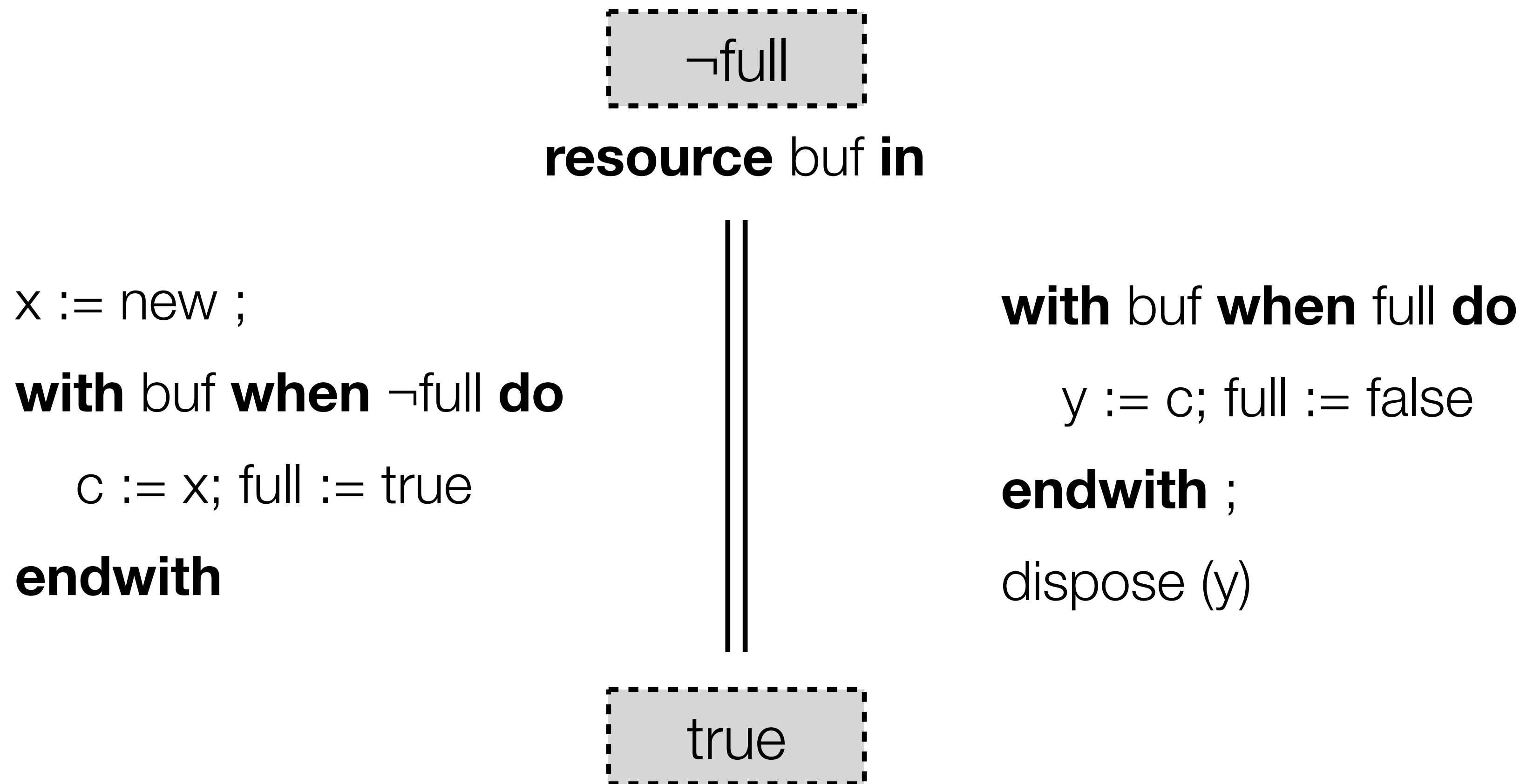
Resource Invariant: sorted(p)

true	‖	true
with r **do** insert(p,7)	‖	**with** r **do** insert(p,5)
true	‖	true
with r **do** insert(p,10)	‖	**with** r **do** remove(p,10)
true	‖	true

end

sorted(p)

Pointer-transferring buffer



Pointer-transferring buffer

Resource Invariant: $\neg full \lor (c \mapsto _ \land full)$

$\{true\}$
x := new ;
$\{x \mapsto _\}$
with buf **when** ¬full **do**
$\{x \mapsto _ \land \neg full\}$
c := x; full := true
$\{c \mapsto _ \land full\}$
endwith
$\{true\}$

||

$\{true\}$
with buf **when** full **do**
$\{c \mapsto _ \land full\}$
y := c; full := false
$\{y \mapsto _ \land \neg full\}$
endwith ;
$\{y \mapsto _\}$
dispose (y)
$\{true\}$