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

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

Disjoint parallelism rule

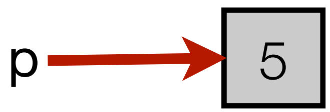
$$\frac{\{\ P_1\ \}\ C_1\ \{\ Q_1\ \} \qquad \{\ P_2\ \}\ C_2\ \{\ Q_2\ \}}{\{\ P_1 \wedge P_2\ \}\ C_1 \parallel C_2\ \{\ Q_1 \wedge Q_2\ \}}\quad \begin{array}{l}\mathrm{fv}(P_1,C_1,Q_1) \cap \mathrm{mod}(C_2) = \varnothing \\ \mathrm{fv}(P_2,C_2,Q_2) \cap \mathrm{mod}(C_1) = \varnothing\end{array}$$

What about programs with pointers?

Part I. Separation logic

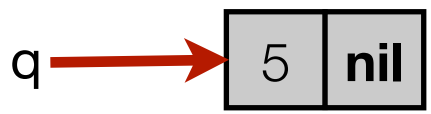
Points-to assertions

SL: convenient syntax for describing the heap



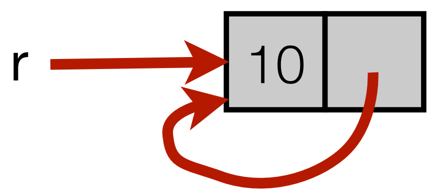
$p \mapsto 5$

heap(p) = 5



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

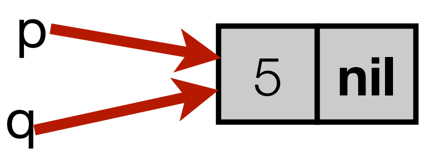
heap(q) = 5 ∧
heap(q+1) = nil



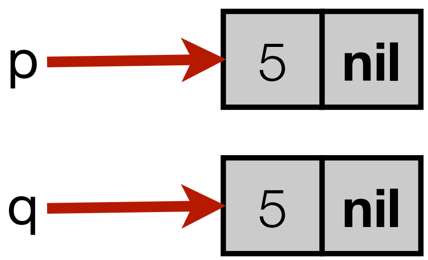
$r \mapsto 10,r$

Conjunction

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



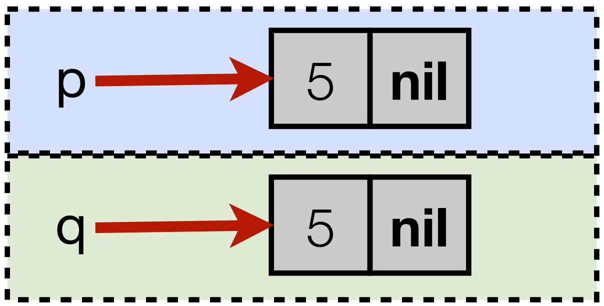
$p \mapsto 5,nil \ \wedge p=q$



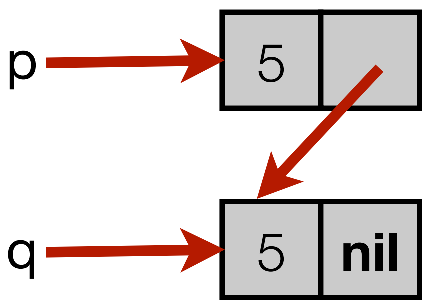
$p \mapsto 5,nil \ \wedge q \mapsto 5,nil \wedge p \neq q$

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

Separating conjunction

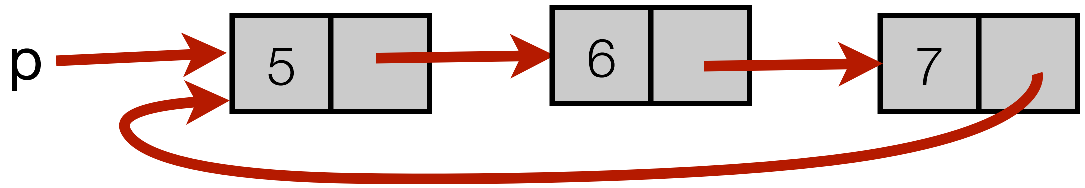


$$\boxed{p \mapsto 5,\mathrm{nil}} * \boxed{q \mapsto 5,\mathrm{nil}}$$



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

A bigger assertion



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

Classical vs intuitionistic SL

Classical SL

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

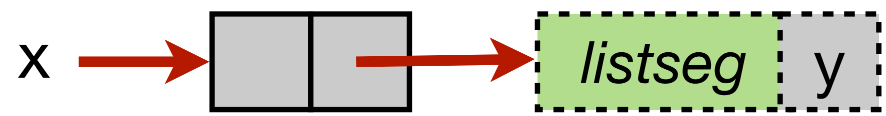
Intuitionistic SL

$p \mapsto 5$	$heap(p) = 5$
emp	true

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

Inductive definitions: list segments

x=y



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

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:

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

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

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

Frame rule

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

Example:

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

Disjoint concurrency

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

Well-specified processes ‘mind their own business’

Parallel merge sort

$\boxed{\text{list}(p)}$

```
mergesort (p) {

   ...

}
```

$\boxed{\text{sorted}(p)}$

$\boxed{\{\ \text{list}(p)\ \}\ \textbf{split}(p, q)\ \ \{\ \text{list}(p) * \text{list}(q)\ \}}$

$\boxed{\{\ \text{sorted}(p) * \text{sorted}(q)\ \}\ \textbf{merge}(p, q)\ \{\ \text{sorted}(p)\ \}}$

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

$\boxed{\text{list}(p)}$

mergesort (p) { local q; ...

if (...) {

$\boxed{\text{list}(p)}$

split (p, q);

$\boxed{\text{list}(p) * \text{list}(q)}$

mergesort (p) || mergesort (q) ;

$\boxed{\text{sorted}(p) * \text{sorted}(q)}$

merge (p, q);

$\boxed{\text{sorted}(p)}$

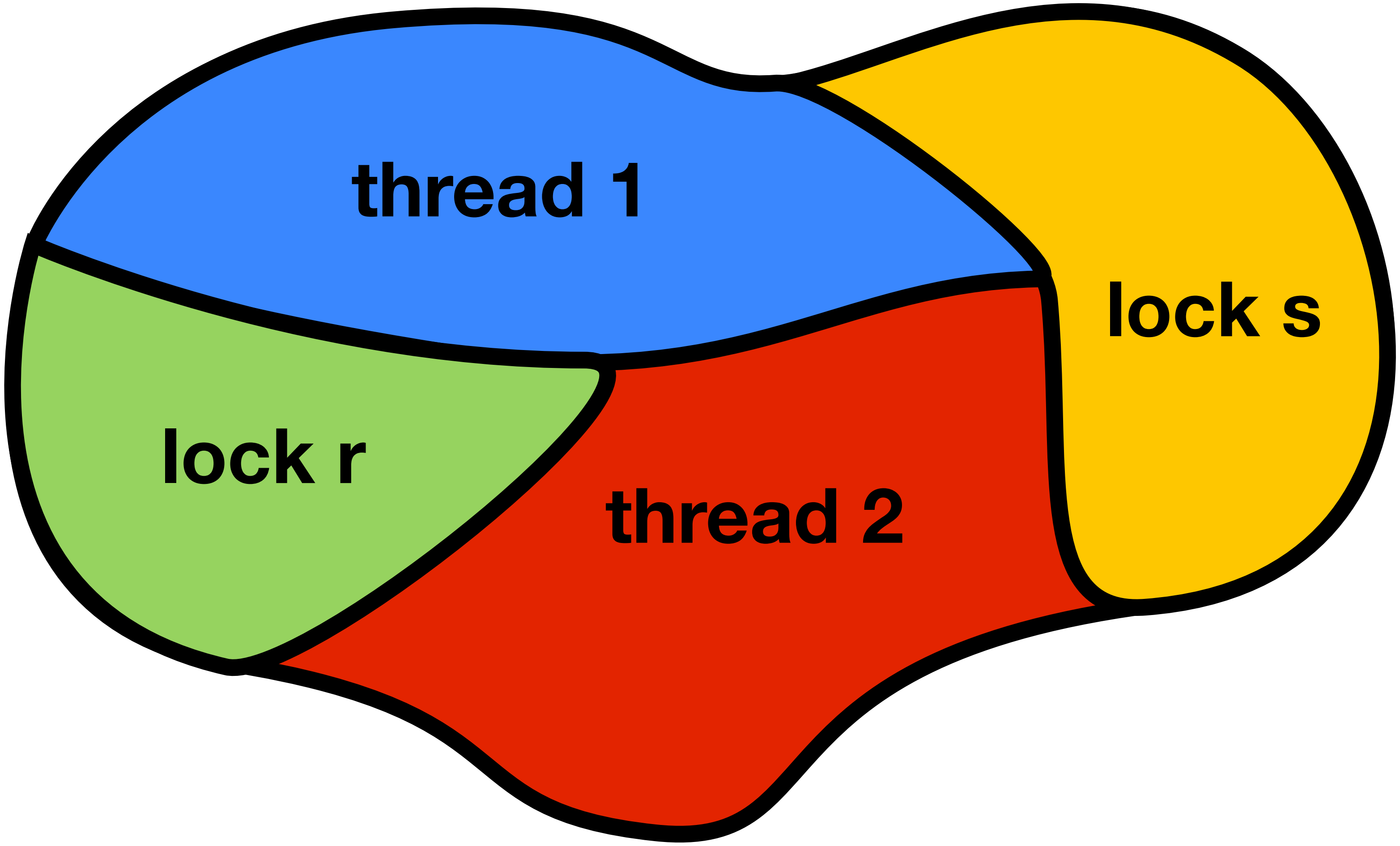
} }

$\boxed{\text{sorted}(p)}$

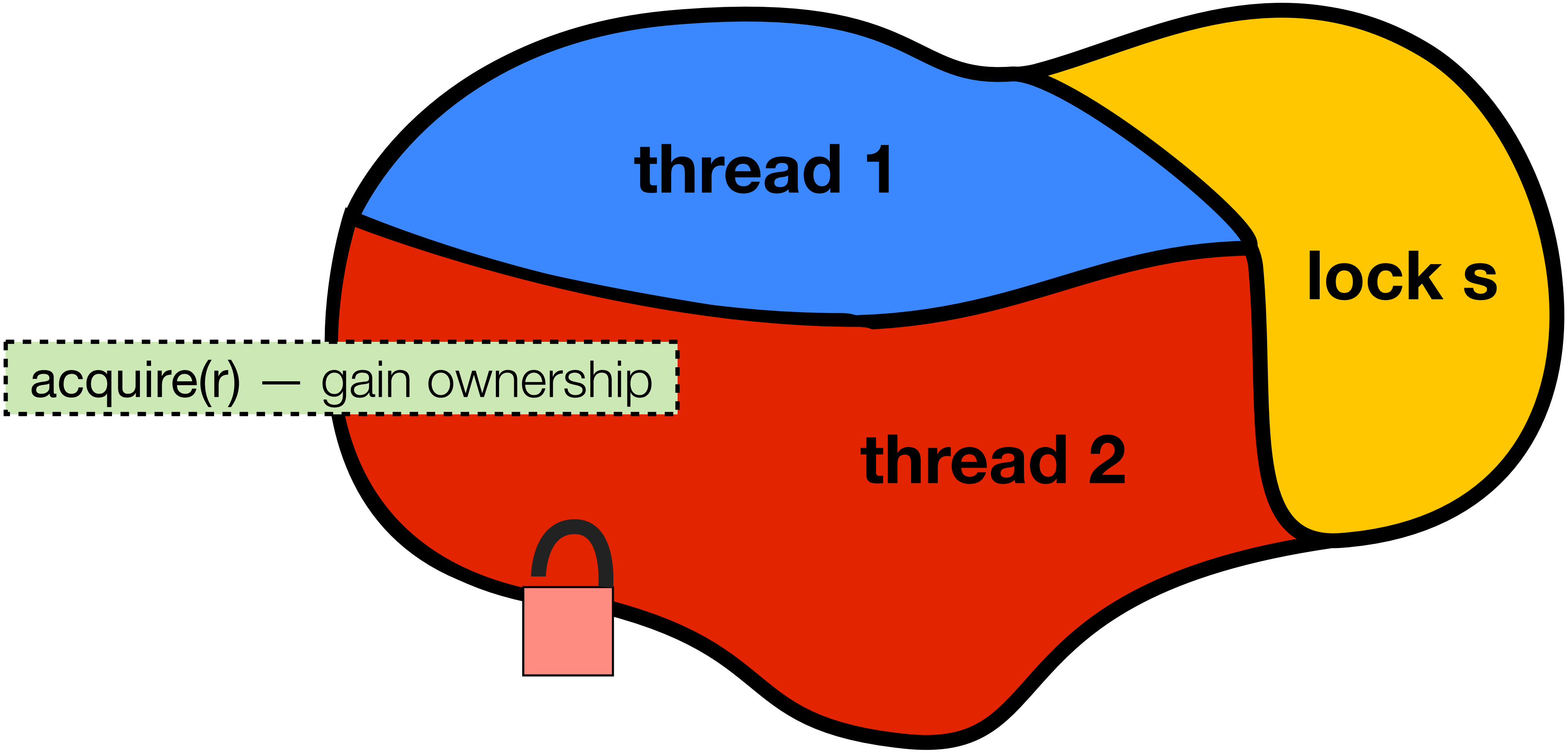
$\boxed{\{ \text{list}(p) \}\ \textbf{split}(p, q)\ \{ \text{list}(p) * \text{list}(q) \}}$

$\boxed{\{ \text{sorted}(p) * \text{sorted}(q) \}\ \textbf{merge}(p, q)\ \{ \text{sorted}(p) \}}$

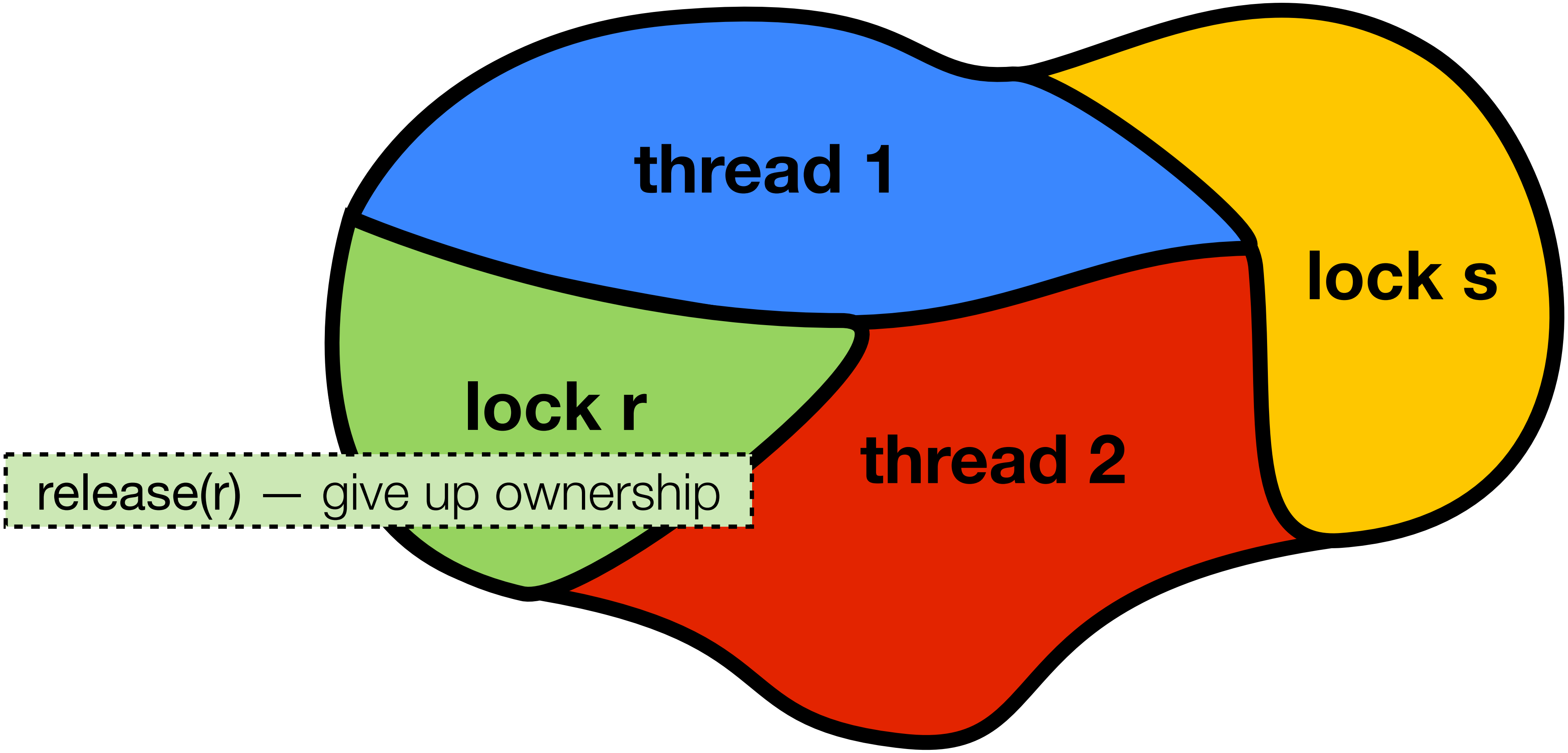
Resource invariants & ownership transfer



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()	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

||

true

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

true

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

true

end

sorted(p)

Pointer-transferring buffer

$\boxed{\neg\text{full}}$

resource buf **in**

x := new ;

with buf **when** ¬full **do**

  c := x; full := true

endwith

||

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

  y := c; full := false

endwith ;

dispose (y)

$\boxed{\text{true}}$

Pointer-transferring buffer

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

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

$\|$

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

Tools for separation logic

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

- User writes specs & proof (using tactics)

Stand-alone program verifiers: Smallfoot, Hip, Verifast, Jstar

- User writes specs & loop invariants

Shape analyses: Space Invader, Thor, Xisa, SLAyer

- User writes specs only (or nothing!)

Mostly just sequential programs