Local Rely-Guarantee Reasoning

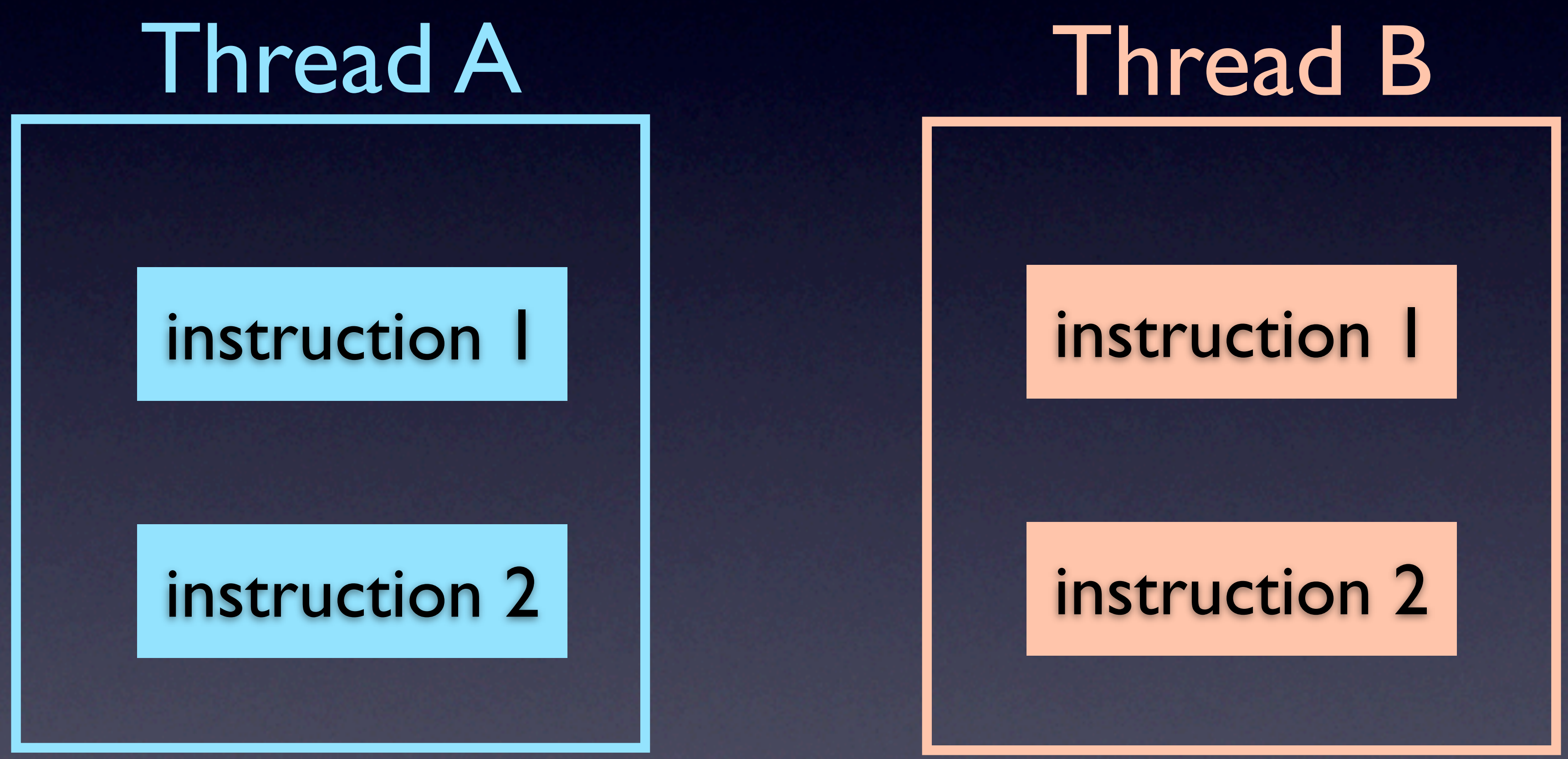
A talk by John Wickerson
after work by Xinyu Feng

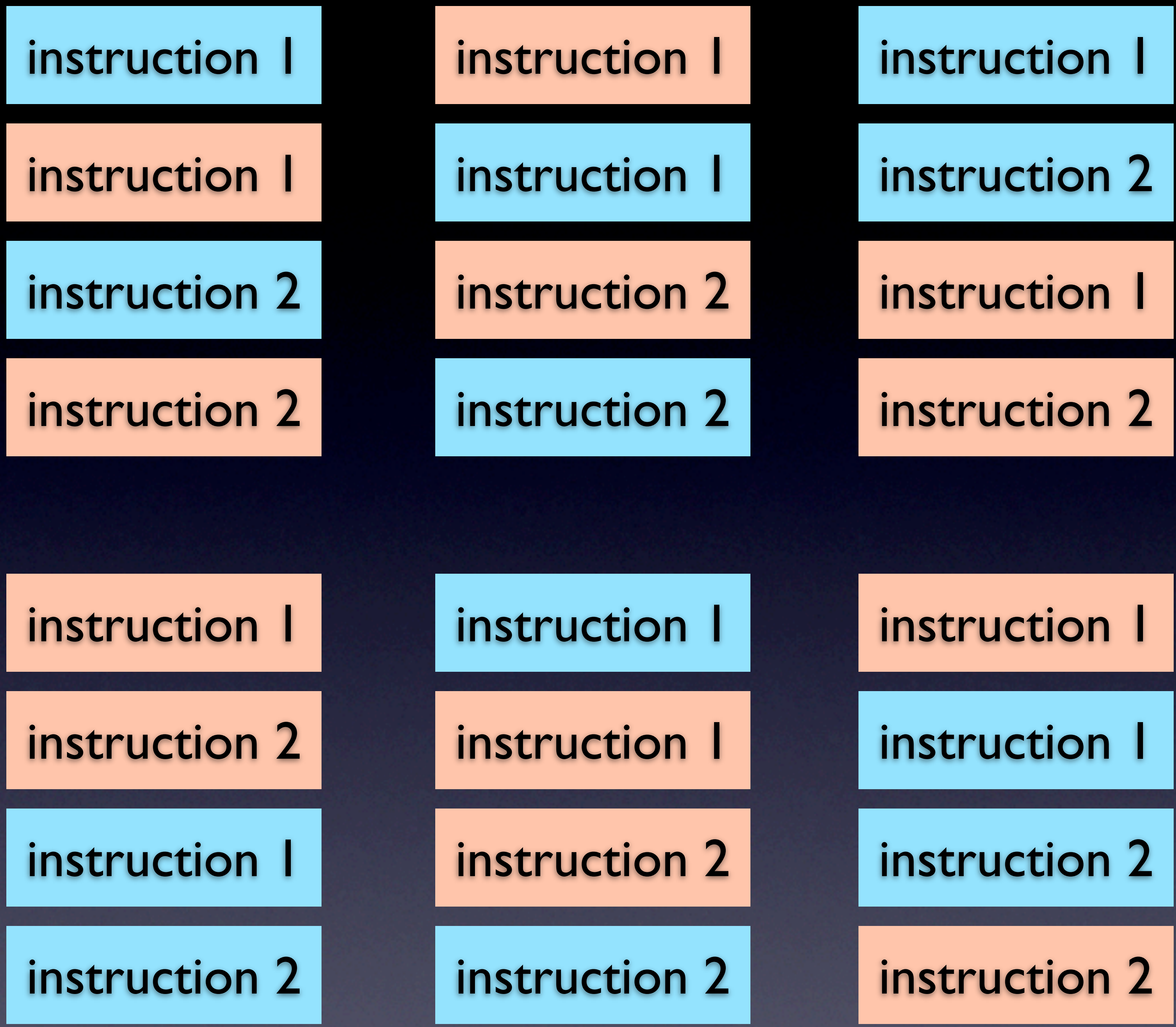
Talk outline

- The RG method and its limitations
- Introduction to LRG
- Formal treatment
- LRG proof rules

The RG method and its limitations

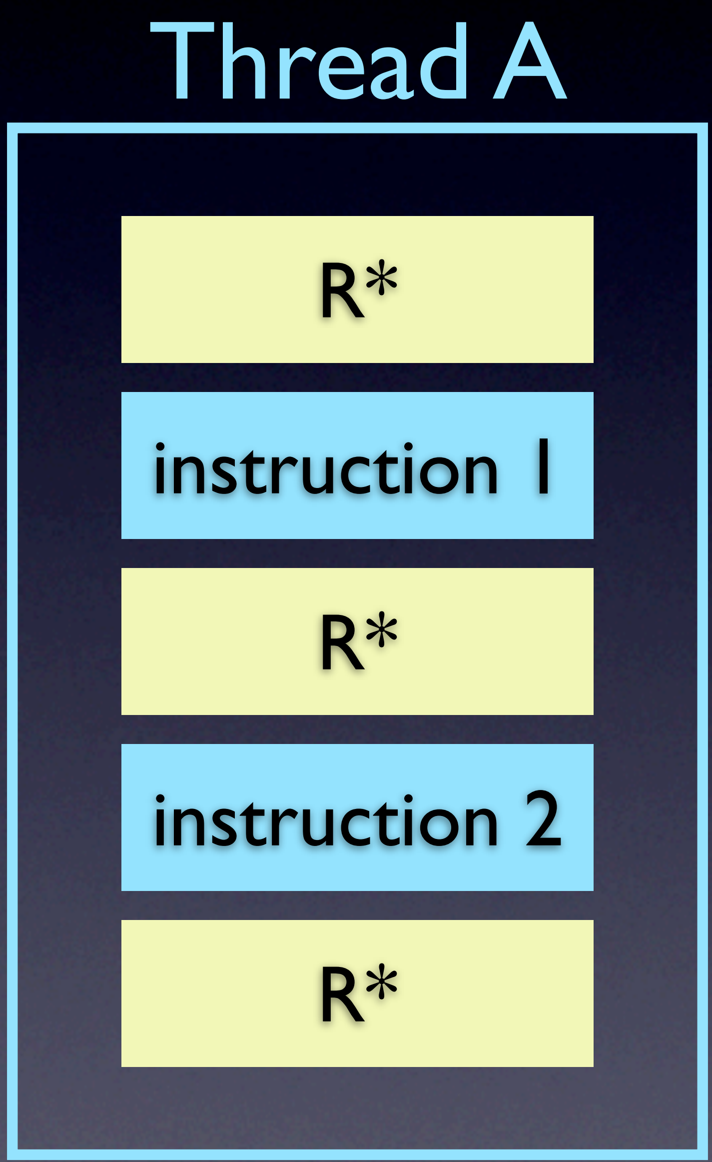
Concurrent programs





Number of possible interleavings of n threads, each executing k instructions, is roughly n^{nk}

Rely-guarantee



R* =

any state transition
that can be done by
any other thread, repeated
zero or more times

Mention that assertions must be stable, which means their validity must be preserved by R*

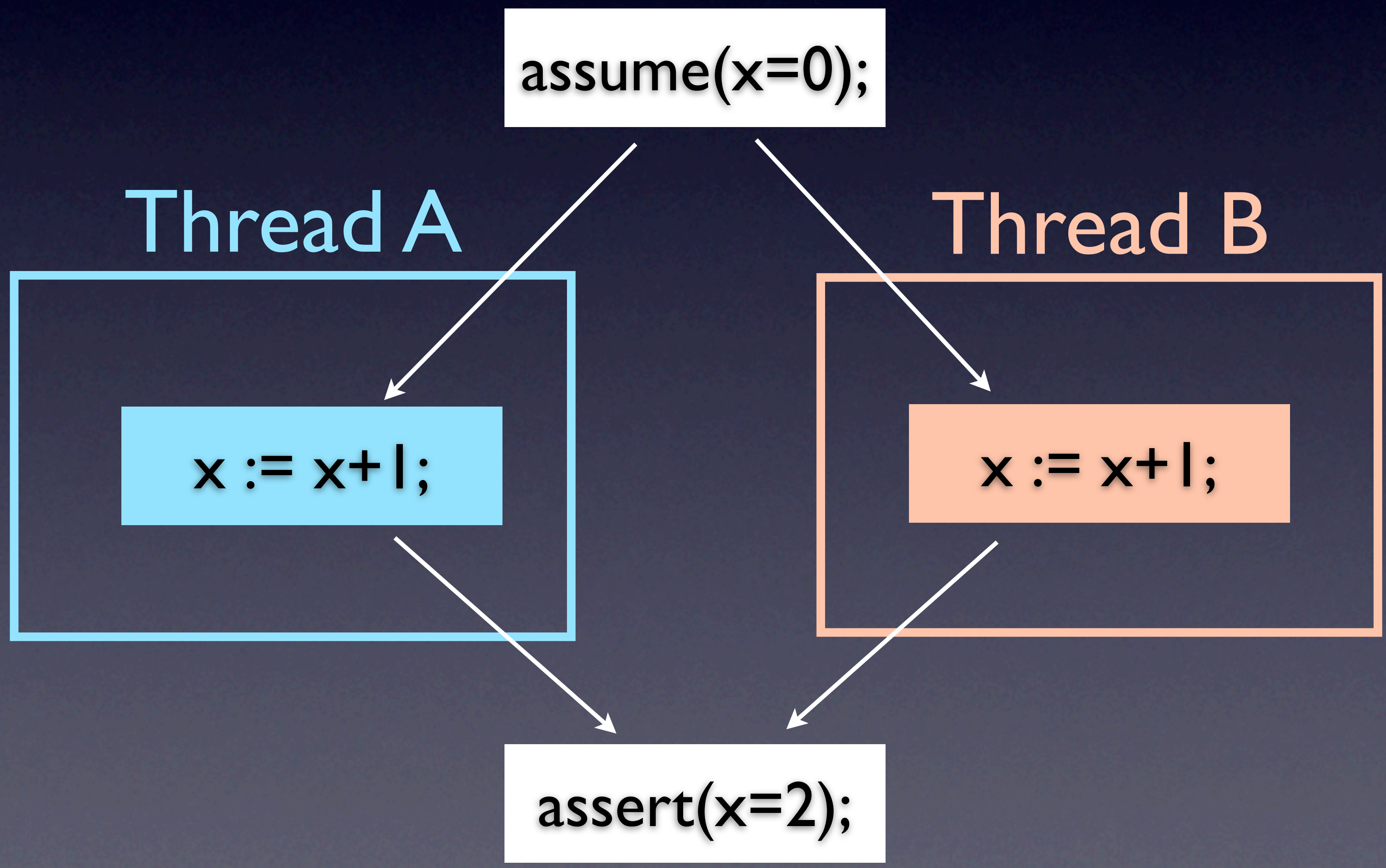
The RG abstraction

- Forget:
  - which thread performs the action
  - in what order the actions are performed
  - how many times the action is performed
- Usually, this is fine...

The action may be performed once, a million times, or not at all.

The RG abstraction

- ...but sometimes too coarse:



No method yet for verifying this program without adding auxiliary state.

$R, G \models \{p\}\ C\ \{q\}$

provided:

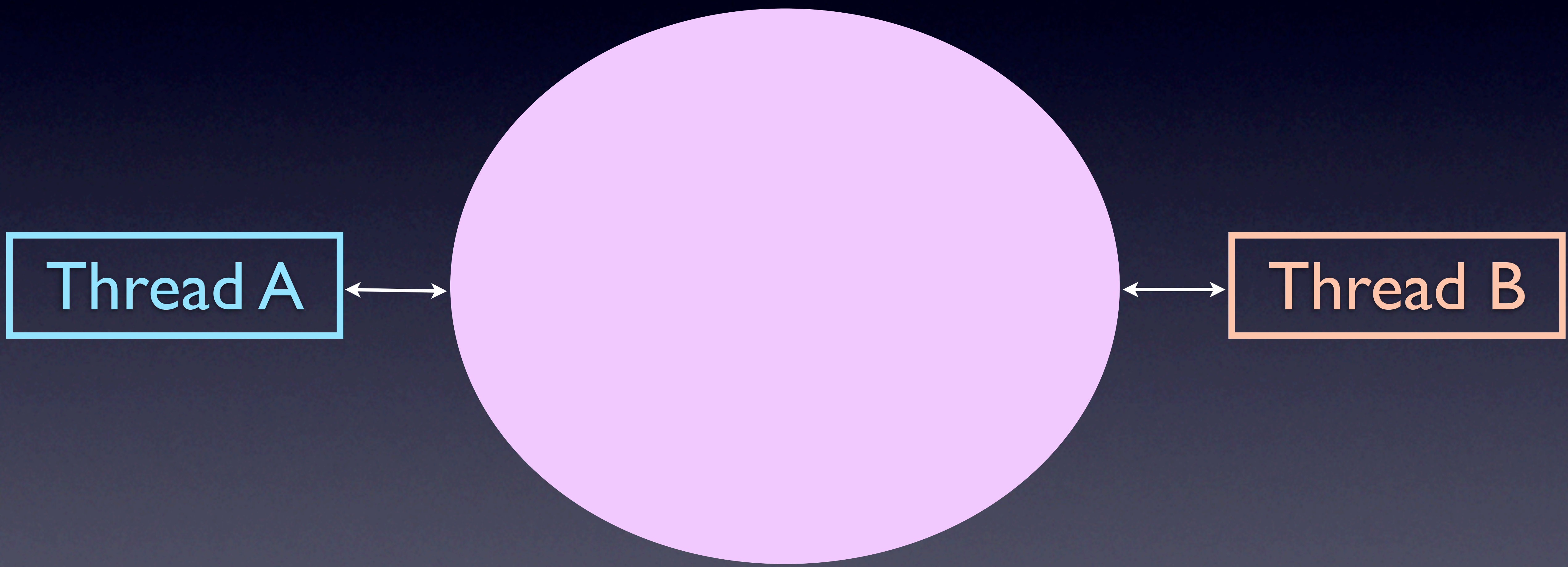
- execution of C begins in a state satisfying p
- environmental transitions are limited to those in R

then:

- any transitions made by C will be within G
- if C terminates it will do so in a state satisfying q

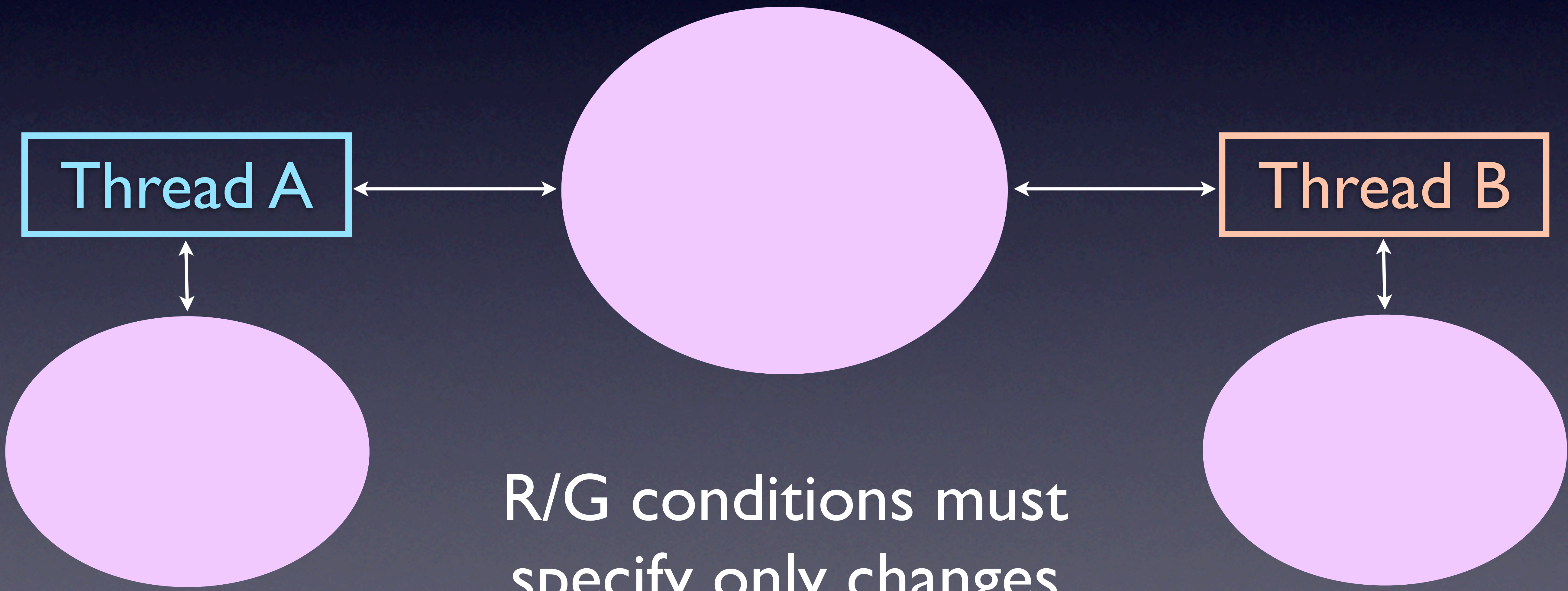
Semantics of RG judgement. Still applicable for LRG, but R and G will be slightly different objects.

RG state model



R/G conditions must specify **all** changes to the state

RGSep state model

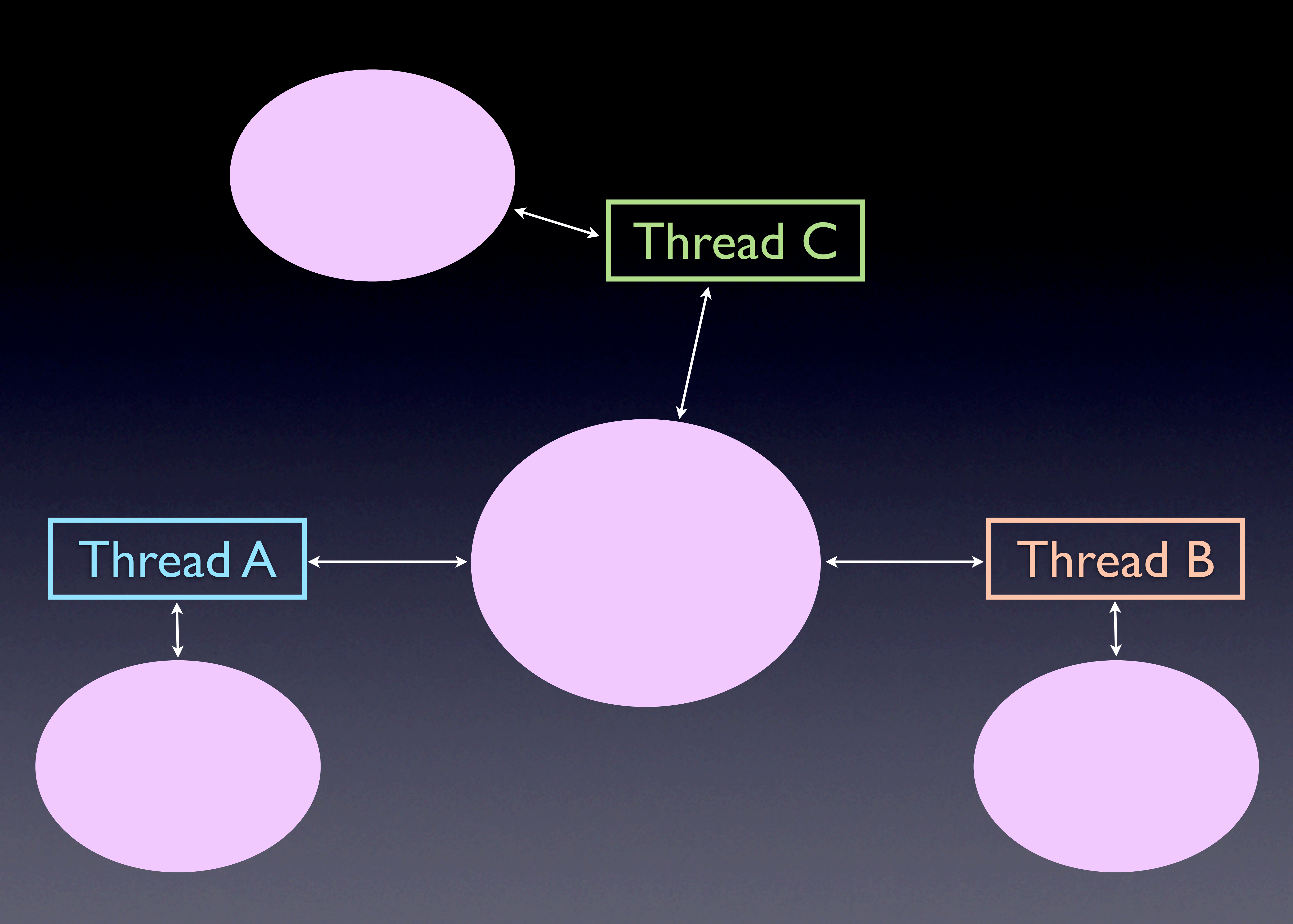


R/G conditions must specify only changes to the **shared** state

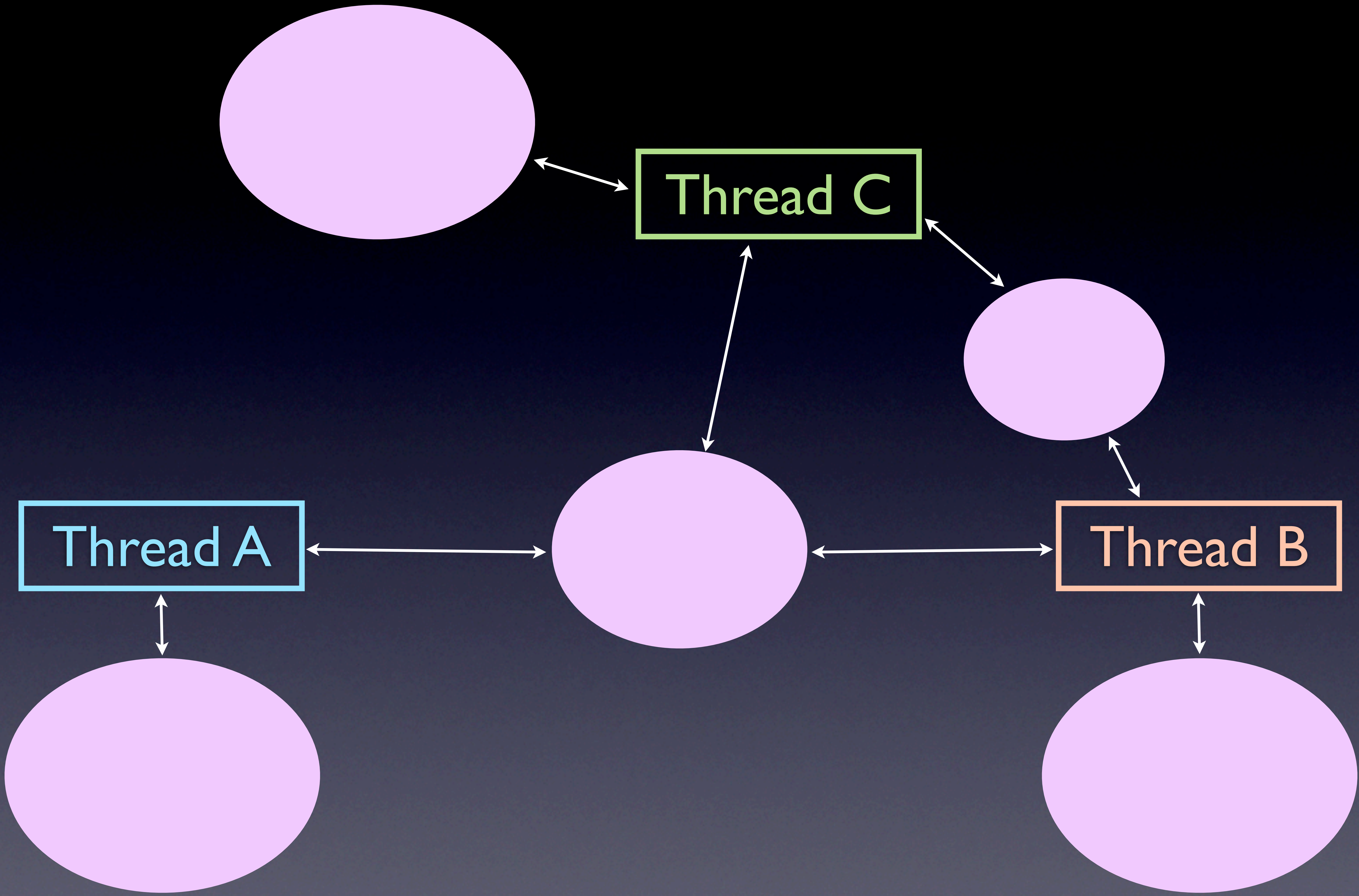
Used by RGSep. Use separation logic to describe each statelet. Still quite coarse model though.

RGSep state model

- Still quite coarse. State is either local or shared between **all** threads



Suppose we have another thread, C...



... we can't talk of 'the state shared between just B and C'. Compare with RGSep's multiple regions – can have multiple regions of shared state, but they're all globally shared.

RGSep state model

- Still quite coarse. State is either local or shared between **all** threads
- Shared resource must be globally known. Makes dynamic allocation of shared resource difficult

RGSep state model

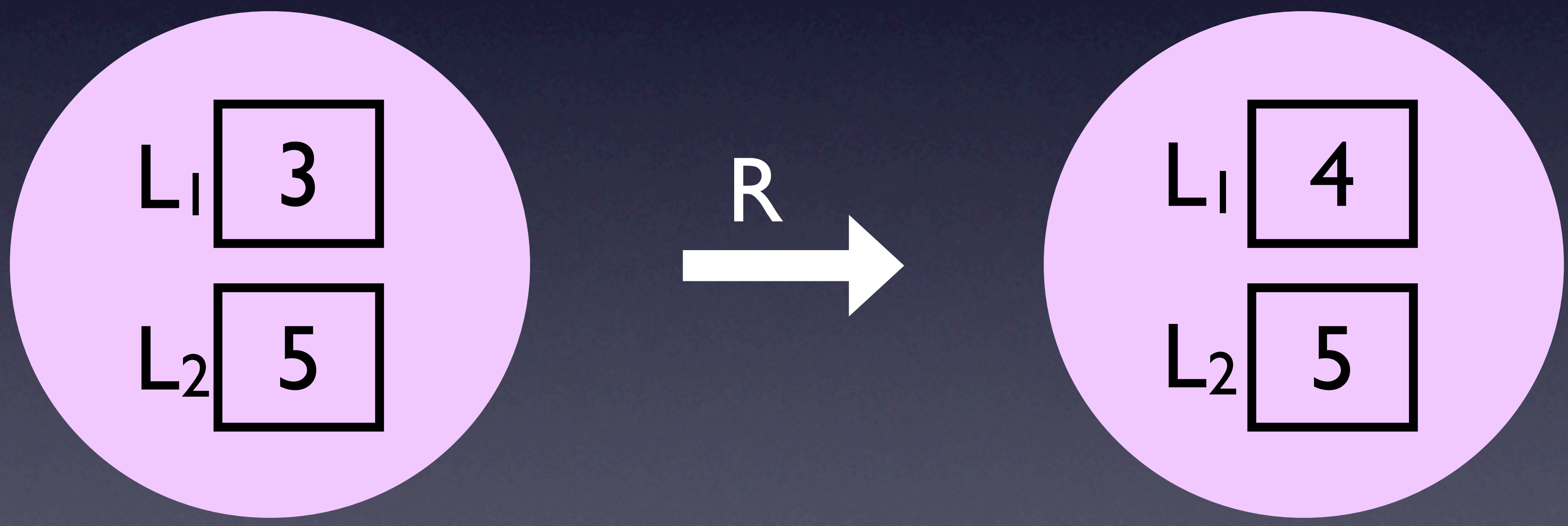
- Still quite coarse. State is either local or shared between **all** threads
- Shared resource must be globally known. Makes dynamic allocation of shared resource difficult
- Hard to make reusable specifications of modules

when specifying a module, the rely and guarantee must mention the entire shared state, even if the module accesses only part of it. Limits reuse of that specification in a different (e.g. larger) shared state.

Introduction to LRG

RGSep actions

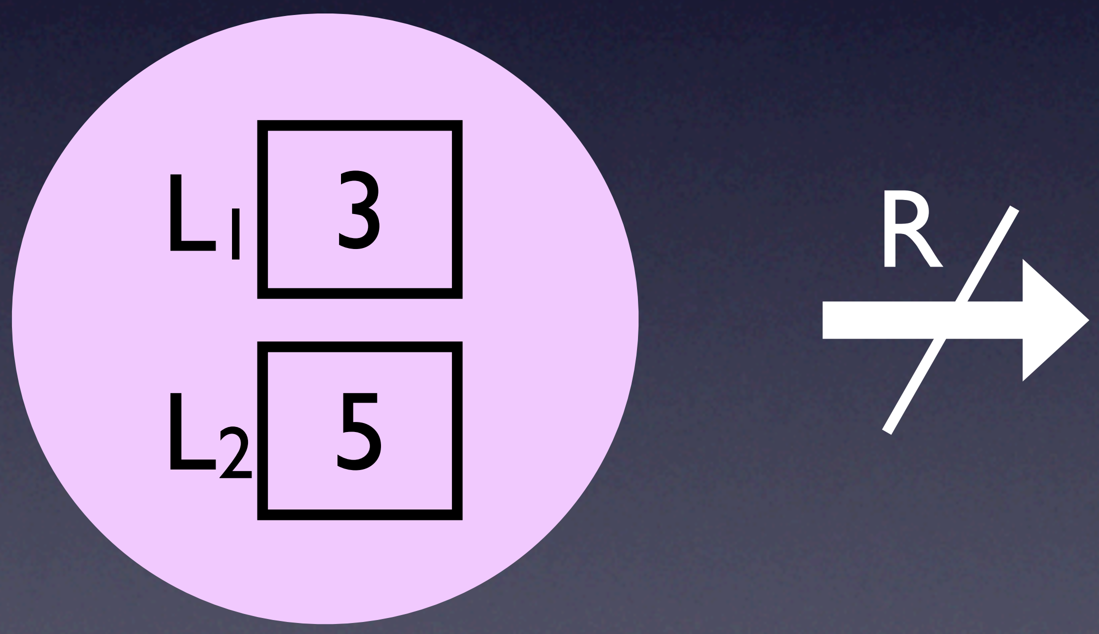
$$R = \{ (L_1 \mapsto 3) \rightsquigarrow (L_1 \mapsto 4) \}$$



Action fires on a state if precondition describes *part* of it.

LRG actions

$$R = \{ (L_1 \mapsto 3) \rightsquigarrow (L_1 \mapsto 4) \}$$



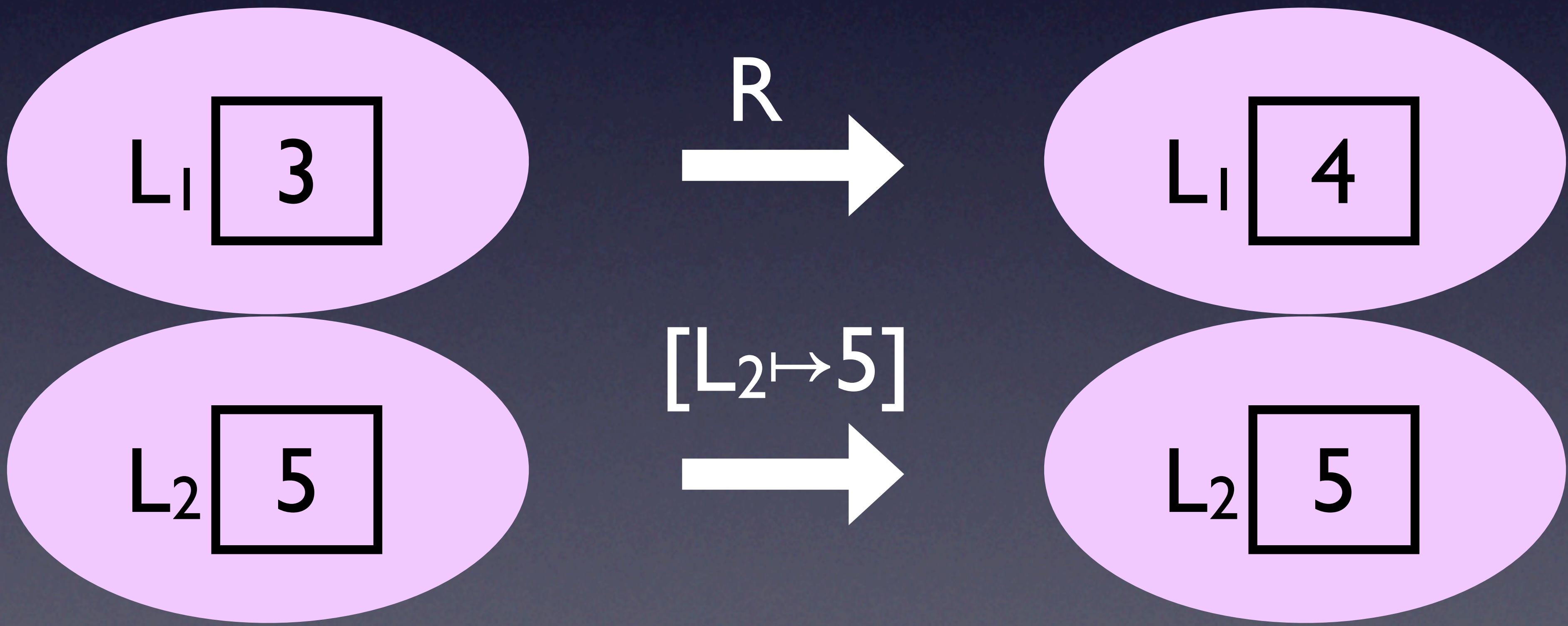
Action only fires on a state described *fully* by the precondition.

LRG actions

$$R = \{ (L_1 \mapsto 3) \rightsquigarrow (L_1 \mapsto 4) \}$$

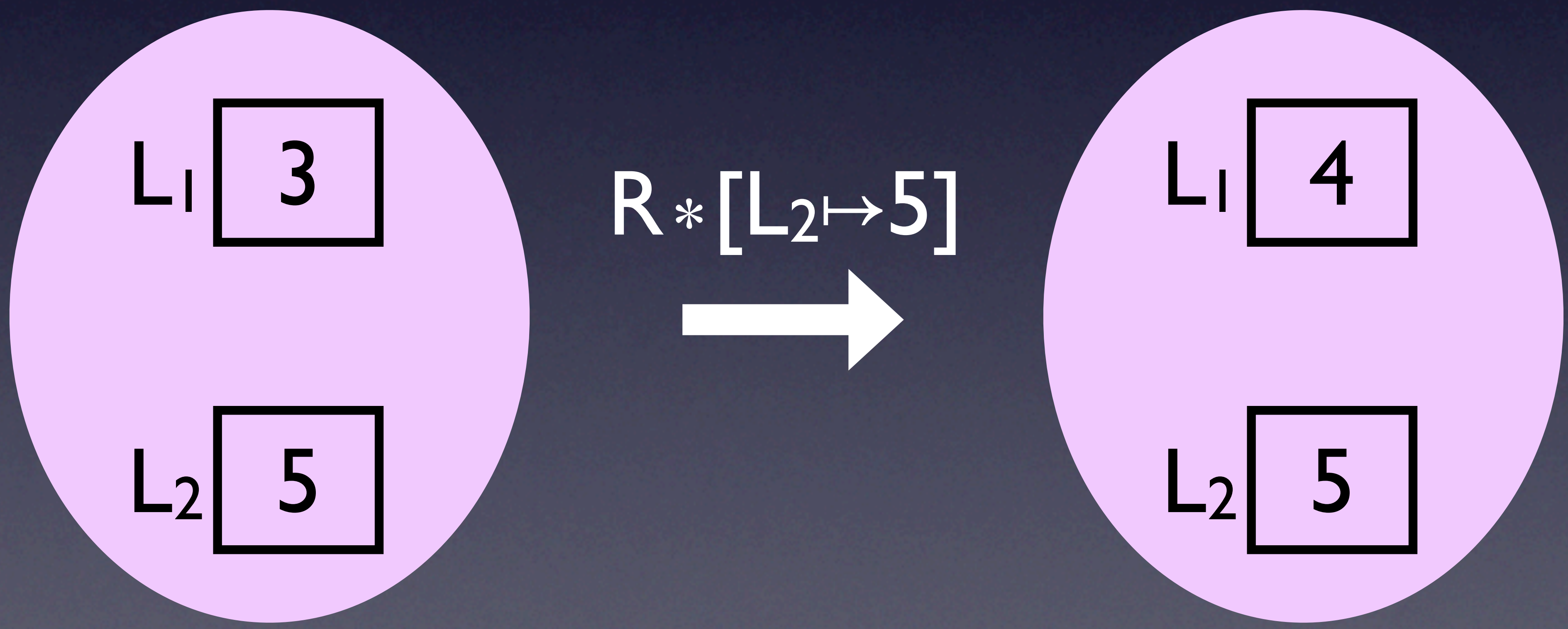
LRG actions

$R = \{\ (L_1 \mapsto 3) \rightsquigarrow (L_1 \mapsto 4)\ \}$



LRG actions

$$R = \{ (L_1 \mapsto 3) \rightsquigarrow (L_1 \mapsto 4) \}$$



We’ve starred together actions, just like in separation logic. As in the spirit of separation logic, what we’re going to be able to do is define small actions, that act only on that part of the state that we need, and then ‘frame in’ other actions that affect other parts of the state.

Specifications of modules will thus specify only the small actions, and then let other actions be framed in when the module is put into a particular context.

Formal treatment

Programming language

commands, C ::=	c	C*
	skip	C ; C
	C + C	C || C

- Commands affect both the **store** and the **heap**
- Basic commands are just elements of
  (store × heap) × (store × heap)

Using a generic programming language for simplicity. Discarding ‘variables as resource’, so store contains both program variables and logical variables.

Assertion language

assertions, p ::=

true	$E \mapsto E$
false	$\exists X.\ p$
$E = E$	$p \wedge p$
$E > E$	$p \vee p$
emp	$p * p$

How we describe the state. (E is pure expression.)

Semantics of assertions

$$
\begin{array}{rcl}
\sigma \models \mathrm{true} & \Leftrightarrow & \text{always} \\
\sigma \models \mathrm{false} & \Leftrightarrow & \text{never} \\
(s,h) \models E_1 = E_2 & \Leftrightarrow & [\![E_1]\!]_s = [\![E_2]\!]_s \\
(s,h) \models E_1 > E_2 & \Leftrightarrow & [\![E_1]\!]_s > [\![E_2]\!]_s \\
(s,h) \models \mathrm{emp} & \Leftrightarrow & h = \{\ \} \\
(s,h) \models E_1 \mapsto E_2 & \Leftrightarrow & h = \{[\![E_1]\!]_s \mapsto [\![E_2]\!]_s\} \\
(s,h) \models \exists X.\ p & \Leftrightarrow & \exists v.\ (s \uplus \{X \mapsto v\}, h) \models p \\
\sigma \models p_1 \wedge p_2 & \Leftrightarrow & \sigma \models p_1 \text{ and } \sigma \models p_2 \\
\sigma \models p_1 \vee p_2 & \Leftrightarrow & \sigma \models p_1 \text{ or } \sigma \models p_2 \\
(s,h) \models p_1 * p_2 & \Leftrightarrow & h = h_1 \uplus h_2 \text{ and } (s,h_1) \models p_1 \text{ and } (s,h_2) \models p_2
\end{array}
$$

Action language

$$\text{actions}, A ::= \quad p \rightsquigarrow p \qquad A \wedge A$$
$$[p] \qquad A \vee A$$
$$\exists X.\, A \qquad A * A$$

Common actions:

$$\begin{aligned} \text{Emp} &= \text{emp} \rightsquigarrow \text{emp} \\ \text{True} &= \text{true} \rightsquigarrow \text{true} \\ \text{Id} &= [\text{true}] \end{aligned}$$

Semantics of actions

$((s,h), (s,h')) \models p_1 \rightsquigarrow p_2 \;\Leftrightarrow\; (s,h) \models p_1$ and $(s,h') \models p_2$

$(\sigma, \sigma) \models [p] \;\Leftrightarrow\; \sigma \models p$

$((s,h), (s,h')) \models \exists X.A \;\Leftrightarrow\; \exists v.\ ((s \uplus \{X \mapsto v\}, h), (s \uplus \{X \mapsto v\}, h')) \models A$

$(\sigma, \sigma') \models A_1 \wedge A_2 \;\Leftrightarrow\; (\sigma, \sigma') \models A_1$ and $(\sigma, \sigma') \models A_2$

$(\sigma, \sigma') \models A_1 \vee A_2 \;\Leftrightarrow\; (\sigma, \sigma') \models A_1$ or $(\sigma, \sigma') \models A_2$

$(\sigma, \sigma') \models A_1 * A_2 \;\Leftrightarrow\; \sigma = \sigma_1 \uplus \sigma_2$ and $\sigma' = \sigma'_1 \uplus \sigma'_2$ and $(\sigma_1, \sigma'_1) \models A_1$ and $(\sigma_2, \sigma'_2) \models A_2$

Note that actions don’t change the store, but they may still depend on it

Stability of assertions

$$p \text{ stab } A \iff \begin{array}{l} \text{if } \sigma \models p \\ \text{and } (\sigma, \sigma') \models A \\ \text{then } \sigma' \models p \end{array}$$

In order to be able to reason about stability, we'd like various properties to hold.

Properties of stability

$$\frac{p_1 \text{ stab } A \qquad p_2 \text{ stab } A}{(p_1 \wedge p_2) \text{ stab } A}$$

$$\frac{p \text{ stab } A_1 \qquad p \text{ stab } A_2}{p \text{ stab } (A_1 \vee A_2)}$$

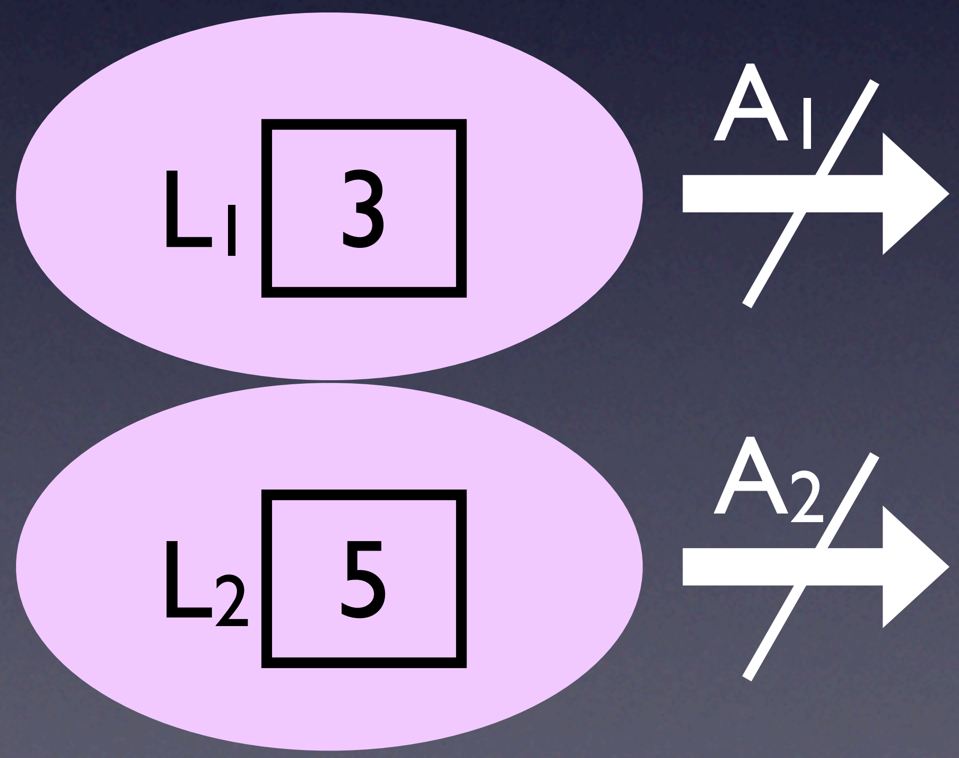
$$\frac{p_1 \text{ stab } A_1 \qquad p_2 \text{ stab } A_2}{(p_1 \vee p_2) \text{ stab } (A_1 \wedge A_2)}$$

$$\frac{p_1 \text{ stab } A_1 \qquad p_2 \text{ stab } A_2}{(p_1 * p_2) \text{ stab } (A_1 * A_2)}$$

Stability problem

$p_1 = L_1 \mapsto 3 \qquad A_1 = \{\ (L_2 \mapsto 5) \leadsto (L_2 \mapsto 6)\ \}$

$p_2 = L_2 \mapsto 5 \qquad A_2 = \{\ (L_1 \mapsto 3) \leadsto (L_1 \mapsto 4)\ \}$

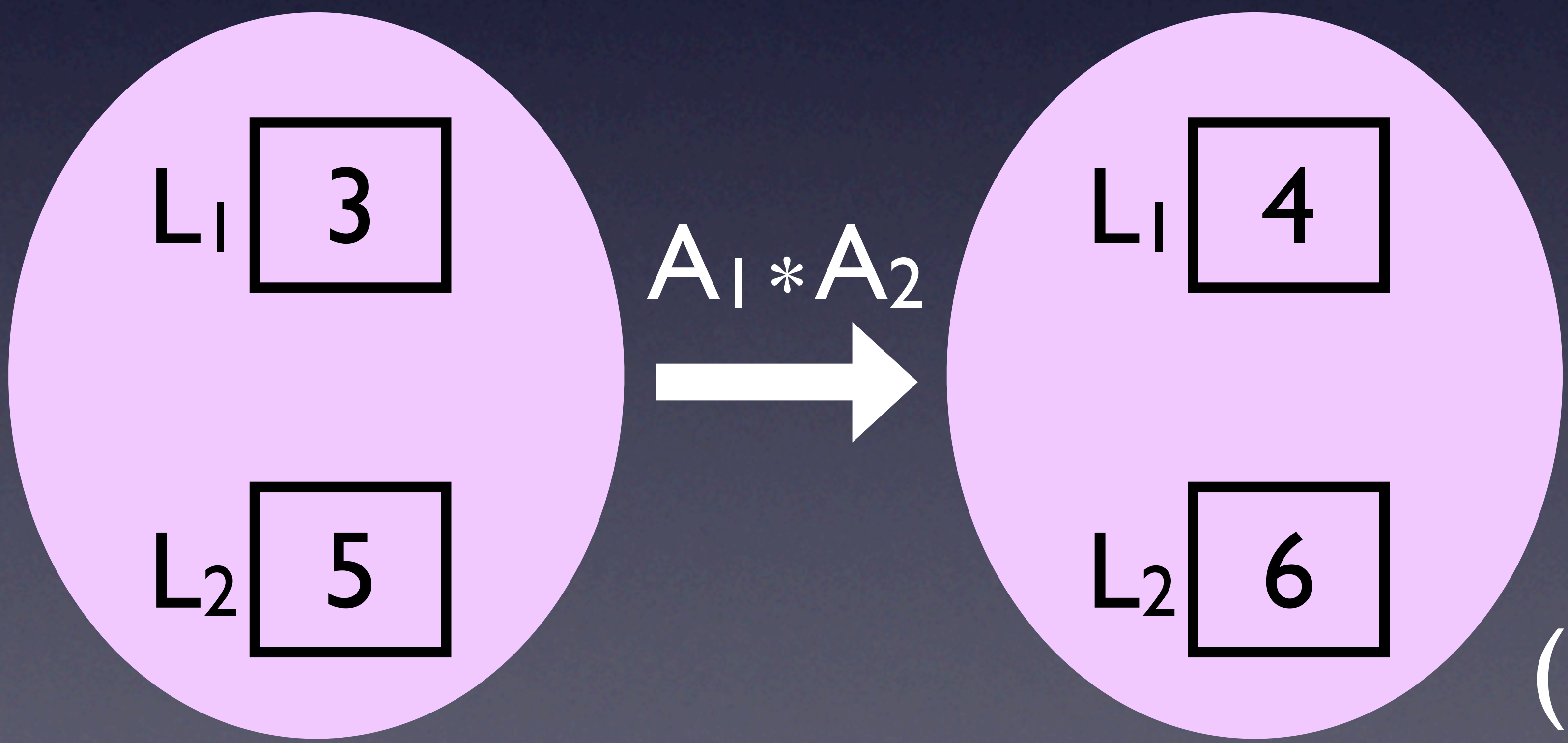


So p_1 stab A_1
and p_2 stab A_2
do hold...

Stability problem

$p_1 = L_1 \mapsto 3$ $\qquad A_1 = \{ (L_2 \mapsto 5) \rightsquigarrow (L_2 \mapsto 6) \}$

$p_2 = L_2 \mapsto 5$ $\qquad A_2 = \{ (L_1 \mapsto 3) \rightsquigarrow (L_1 \mapsto 4) \}$



So p_1 stab A_1
and p_2 stab A_2
do hold...
...but
$(p_1 * p_2)$ stab $(A_1 * A_2)$
does **not** hold

Note that all the assertions are precise, and even that doesn't solve the problem. The problem is that "p stab A" holds vacuously if p and A talk about different parts of the state. Need some way to 'link' them.

Properties of stability

$$\frac{p_1 \text{ stab } A \qquad p_2 \text{ stab } A}{(p_1 \wedge p_2) \text{ stab } A}$$

$$\frac{p \text{ stab } A_1 \qquad p \text{ stab } A_2}{p \text{ stab } (A_1 \vee A_2)}$$

$$\frac{p_1 \text{ stab } A_1 \qquad p_2 \text{ stab } A_2}{(p_1 \vee p_2) \text{ stab } (A_1 \wedge A_2)}$$

$$\frac{p_1 \text{ stab } A_1 \qquad p_2 \text{ stab } A_2}{(p_1 * p_2) \text{ stab } (A_1 * A_2)}$$

✗

Properties of stability

$$\frac{p_1 \text{ stab } A \qquad p_2 \text{ stab } A}{(p_1 \wedge p_2) \text{ stab } A}$$

$$\frac{p \text{ stab } A_1 \qquad p \text{ stab } A_2}{p \text{ stab } (A_1 \vee A_2)}$$

$$\frac{p_1 \text{ stab } A_1 \qquad p_2 \text{ stab } A_2}{(p_1 \vee p_2) \text{ stab } (A_1 \wedge A_2)}$$

$$\frac{\begin{array}{cc} p_1 \Rightarrow i & i \triangleright A_1 \\ p_1 \text{ stab } A_1 & p_2 \text{ stab } A_2 \end{array}}{(p_1 * p_2) \text{ stab } (A_1 * A_2)} \checkmark$$

Invariant-fenced actions

- Invariants 'link' assertions and actions
- $i \rhd A$ means:
  - i is a precise assertion
  - $[i] \Rightarrow A$
  - $A \Rightarrow (i \rightsquigarrow i)$

i is precise   means "of any state at most one substate satisfies i"
[i] => A   means "the action may fire reflexively on any part of the state that satisfies the invariant"
A => (i ~> i)   means "the invariant holds both before and after the action fires"

Invariant-fenced actions

Example. Let

$$A \;=\; \mathrm{List}_m(L) \wedge m \leq n \rightsquigarrow \mathrm{List}_n(L)$$

$$i \;=\; \mathrm{List}(L)$$

Show $i \rhd A$

$$\mathrm{List}_0(x) \;=\; x{=}0 \wedge \mathrm{emp}$$

$$\mathrm{List}_{n+1}(x) \;=\; \exists y.\, x \mapsto y * \mathrm{List}_n(y)$$

$$\mathrm{List}(x) \;=\; \exists n.\, \mathrm{List}_n(x)$$

i is precise because in any heap there is only one way to chase pointers through the heap until you reach the null pointer.

[i] => A because the action allows m=n

A => (i ~> i) because the heap comprises a list from L both before and after the action.

Note that requiring actions to be invariant-fenced doesn’t prohibit them from changing the size of the resource.

LRG proof rules

Proof rules

- Of the form:
  $R, G, i \vdash \{p\}\ C\ \{q\}$
- Well-formedness condition:
  $i \triangleright R$ and $i \triangleright G$ and $p \lor q \Rightarrow i * \mathrm{true}$
- Soundness:
  $R, G, i \vdash \{p\}\ C\ \{q\}$
  $\Rightarrow R * \mathrm{Id}, G * \mathrm{True} \models \{p\}\ C\ \{q\}$

i describes the shared state.

Well-formedness condition is implicit side-condition on all proof rules.

R and G only describe changes to shared state, but p and q include local state, hence *True.

Note that i doesn't feature in the semantics of the judgement.

R and G act only over the shared state; the "overall" rely and guarantee conditions are R*Id (environment cannot do anything to local state) and G*True (we can do anything to our local state).

Proof rules

Basic command

$$\frac{\vdash \{p\}\ c\ \{q\} \qquad p \text{ stab } R * \mathrm{Id} \qquad q \text{ stab } R * \mathrm{Id} \qquad p \rightsquigarrow\!\!\!\!\rightarrow q \Rightarrow G * \mathrm{True}}{R, G, i \vdash \{p\}\ c\ \{q\}}$$

Proof rules

Non-deterministic choice

$$\frac{R, G, i \vdash \{p\}\ C_1\ \{q\} \qquad R, G, i \vdash \{p\}\ C_2\ \{q\}}{R, G, i \vdash \{p\}\ C_1 + C_2\ \{q\}}$$

Proof rules

Non-deterministic looping

$$\frac{\begin{array}{c} R, G, i \vdash \{p\}\ C\ \{p\} \\ p \text{ stab } R * Id \end{array}}{R, G, i \vdash \{p\}\ C^*\ \{p\}}$$

Stability check not actually in paper. Required by case when C* executes as ‘skip’.

Proof rules

Skip

$$\frac{}{\text{Emp, Emp, emp} \vdash \{\text{emp}\}\ \text{skip}\ \{\text{emp}\}}$$

Skip doesn’t change anything, so everything else can be framed in. Vacuously stable.

Proof rules

Sequential composition

$$\frac{R, G, i \vdash \{p\}\ C_1\ \{r\} \qquad R, G, i \vdash \{r\}\ C_2\ \{q\}}{R, G, i \vdash \{p\}\ C_1\ ;\ C_2\ \{q\}}$$

Proof rules

Parallel composition

$$\frac{\begin{array}{c} R \vee G_2, G_1, i \vdash \{p_1 * r\}\ C_1\ \{q_1 * r'\} \\ R \vee G_1, G_2, i \vdash \{p_2 * r\}\ C_2\ \{q_2 * r'\} \\ r \vee r' \Rightarrow i \end{array}}{R, G_1 \vee G_2, i \vdash \{p_1 * p_2 * r\}\ C_1 \parallel C_2\ \{q_1 * q_2 * r'\}}$$

p's and q's describe local states, r's describe shared state.

Proof rules

Hiding

$$\frac{R * R', G * G', i * i' \vdash \{p\}\ C\ \{q\}}{R, G, i \vdash \{p\}\ C\ \{q\}}$$

Allows arbitrary shared state to be claimed as local. Inappropriate hiding detected at point of parallel composition: if the local states are not disjoint from each other and the shared state, then they can't be starred together. Common pattern is to make a bit of shared state for child threads to use.

Proof rules

Frame

$$\frac{R, G, i \vdash \{p\}\ C\ \{q\} \qquad r \text{ stab } R' * Id}{R * R', G * G', i * i' \vdash \{p * r\}\ C\ \{q * r\}}$$

Can frame in shared state or local state.

Proof rules

Weakening

$$\frac{R', G', i' \vdash \{p'\}\ C\ \{q'\} \qquad p \Rightarrow p' \qquad R \Rightarrow R' \qquad G' \Rightarrow G \qquad q' \Rightarrow q}{R, G, i \vdash \{p\}\ C\ \{q\}}$$

Proof rules

Disjunction

$$\frac{R, G, i \vdash \{p_1\}\ C\ \{q_1\} \qquad R, G, i \vdash \{p_2\}\ C\ \{q_2\}}{R, G, i \vdash \{p_1 \lor p_2\}\ C\ \{q_1 \lor q_2\}}$$

Proof rules

Conjunction

$$\frac{R, G, i \vdash \{p_1\}\ C\ \{q_1\} \qquad R, G, i \vdash \{p_2\}\ C\ \{q_2\}}{R, G, i \vdash \{p_1 \land p_2\}\ C\ \{q_1 \land q_2\}}$$

Proof rules

Existential quantification

$$\frac{R, G, i \vdash \{p\}\ C\ \{q\} \qquad x \text{ not free in } R, G \text{ or } i}{R, G, i \vdash \{\exists x.\ p\}\ C\ \{\exists x.\ q\}}$$

Concluding remarks

- Local rely/guarantee conditions
- More refined state model
- Improved ability to reason modularly ...
- ... but precise invariants are restrictive.

Talk about inelegance of equating 'False' with 'Id' in ordinary RG reasoning.

Can modularly verify a multi-threaded module (e.g. ConcurrentGCD in paper)

In CSL, can relax 4th point. Use supported assertions as invariants, and intuitionistic assertions for private state. Can't do this in LRG, for reasons to do with the asymmetry of the rely and guarantee.