RGSep

Lecturer: John Wickerson

Lecture plan

1. Introducing RGSep
2. Verification of a fine-grained concurrent datastructure
3. RG and CSL as special cases
4. Extensions, limitations and further work

Comparison

$J \vdash \{P\}\ C\ \{Q\}$

CONCURRENT SEPARATION LOGIC

uses **invariants**

(state $\rightarrow$ bool)

local reasoning

$R, G \vdash \{P\}\ C\ \{Q\}$

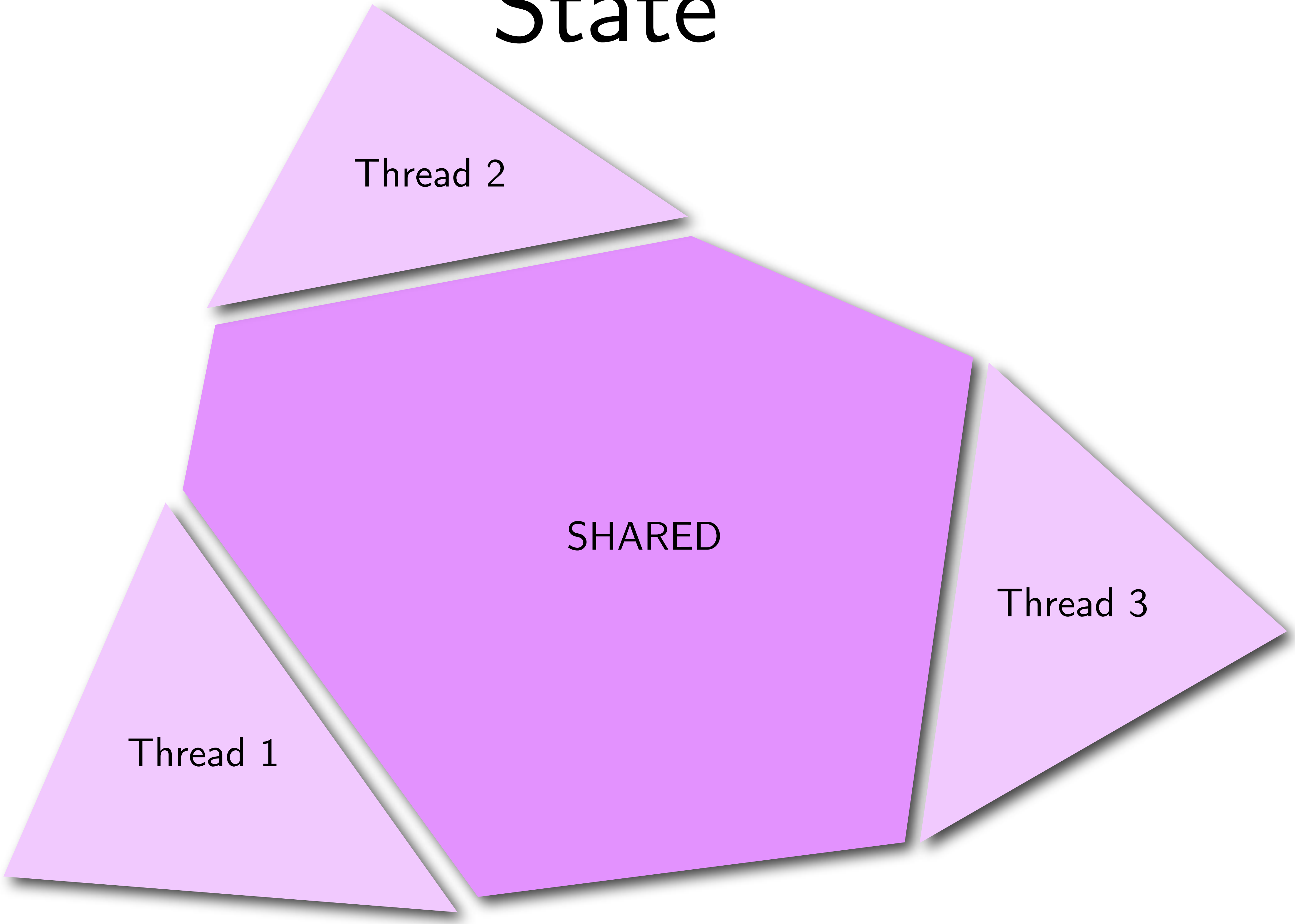
RELY-GUARANTEE

uses **relations**

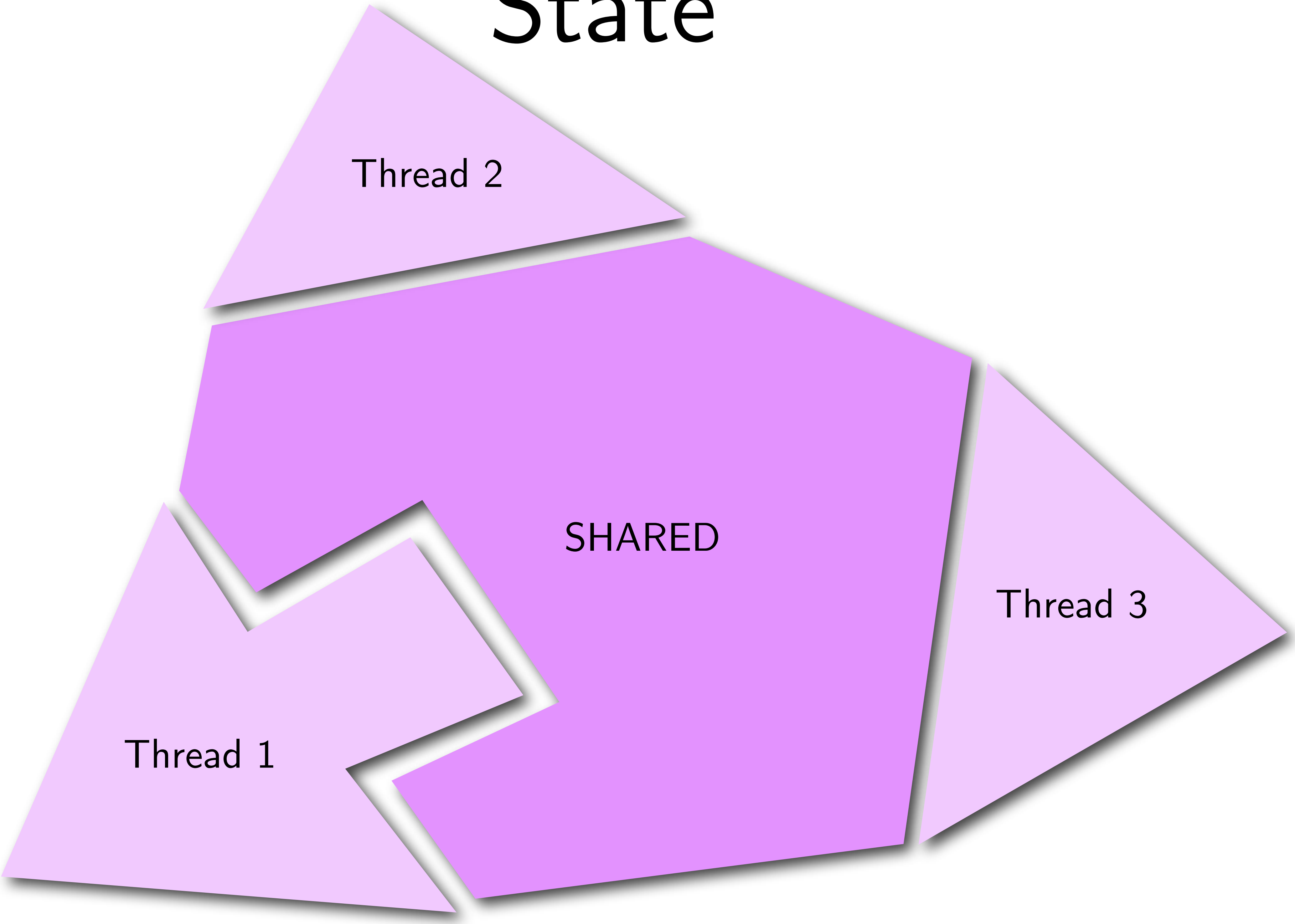
(state $\times$ state $\rightarrow$ bool)

global reasoning

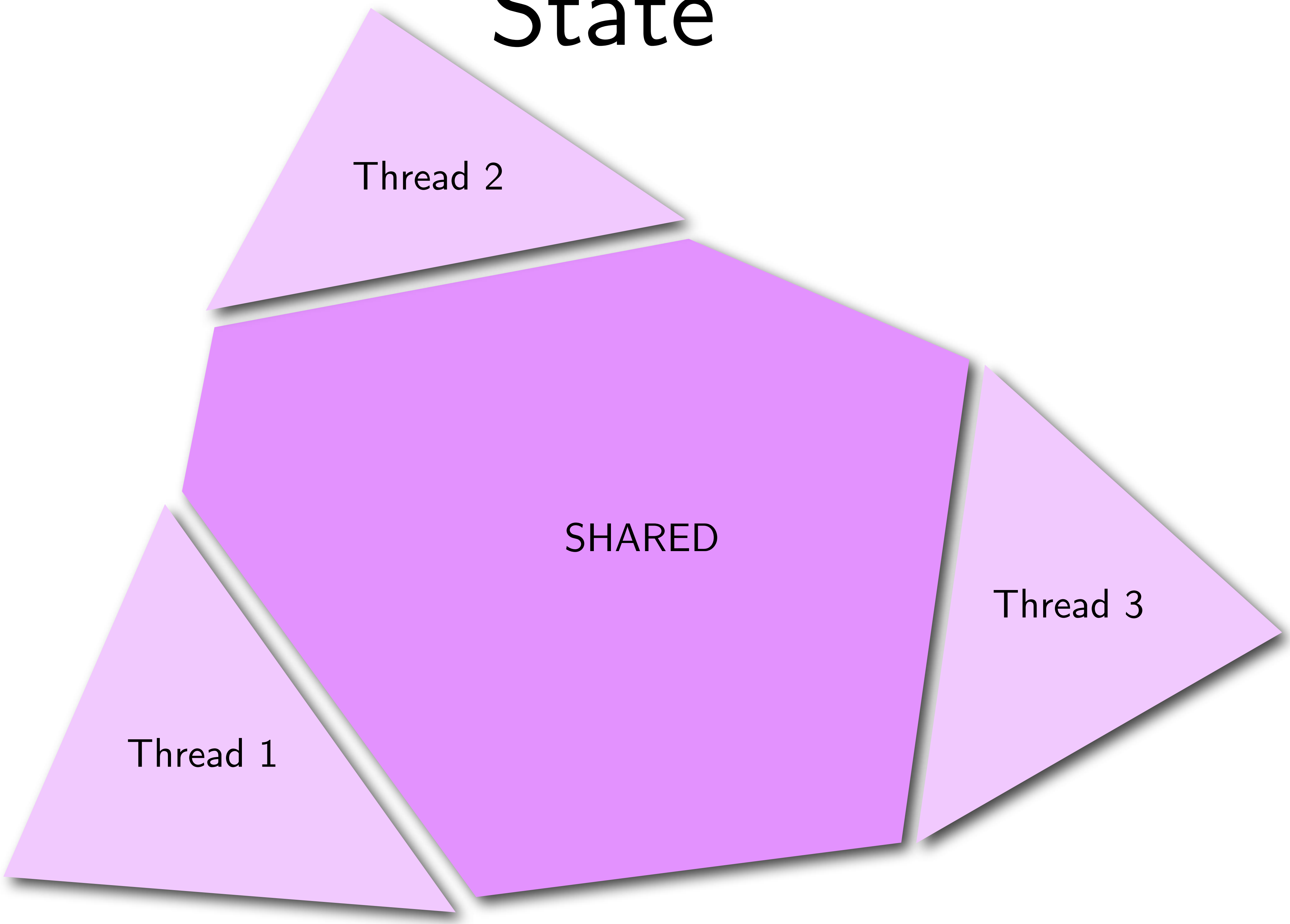
State



State

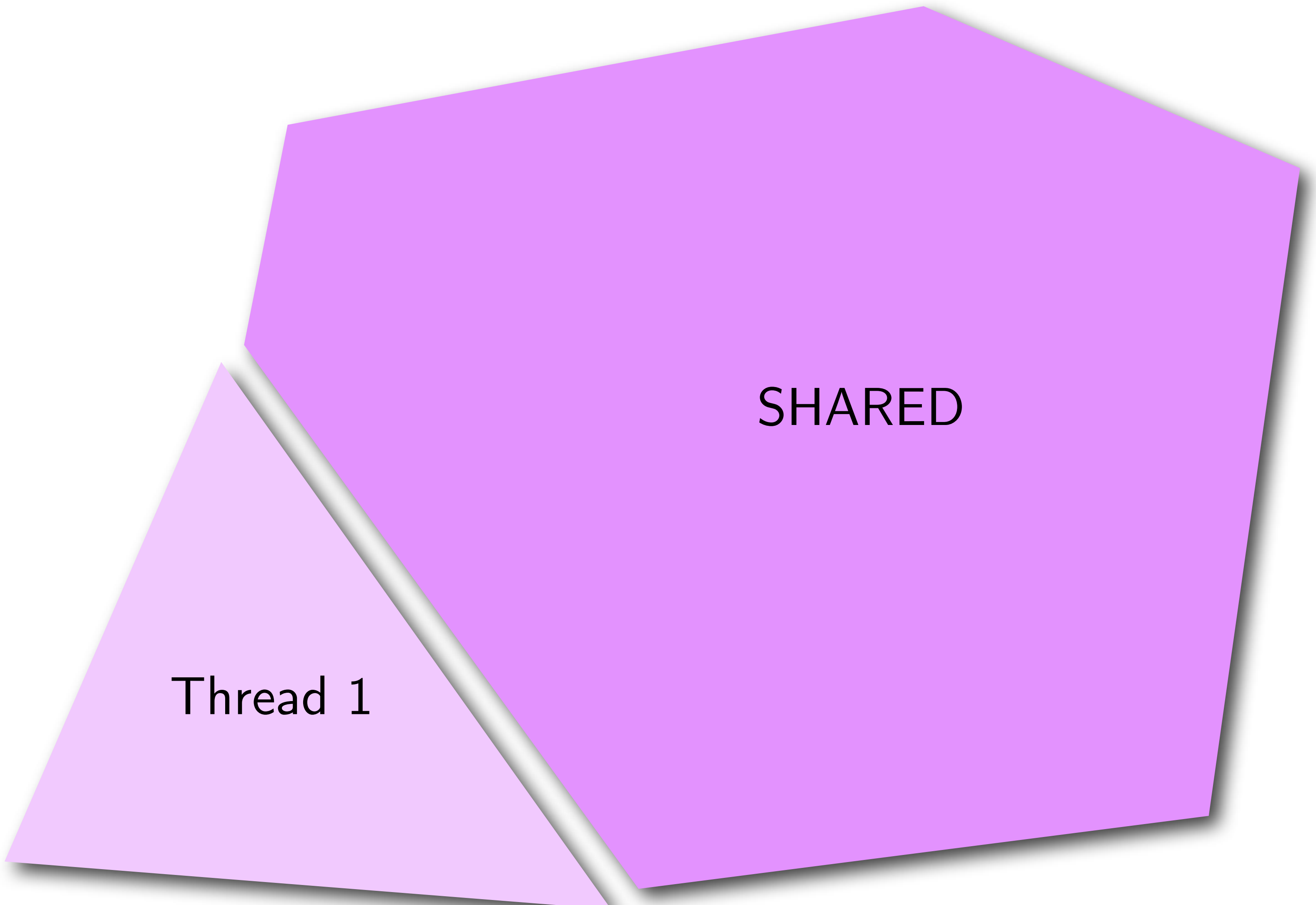


State



State

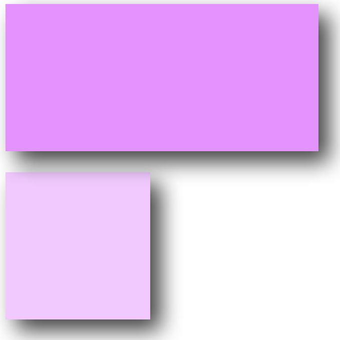
State = { (s,h,H) ∈ Store × Heap × Heap | h ⊥ H }



Assertions

$$P ::= e \mapsto e \mid \mathsf{emp} \mid P * P \mid P \vee P \mid \exists x.P$$

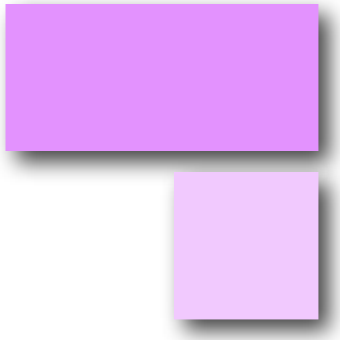
$$p ::= P \mid [P] \mid p * p \mid p \vee p \mid \exists x.p$$



$$\frac{s,h \models P}{s,h,H \models P}$$

$$\frac{s,H \models P \qquad h=\emptyset}{s,h,H \models [P]}$$

$$\frac{s,h_1,H \models p_1 \qquad s,h_2,H \models p_2 \qquad h=h_1 \uplus h_2}{s,h,H \models p_1 * p_2}$$



Assertions

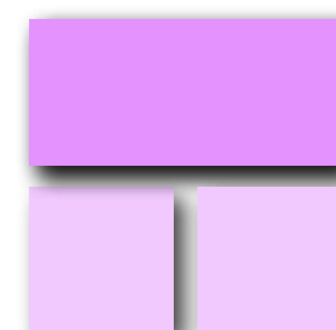
$$P ::= e \mapsto e \mid \mathrm{emp} \mid P * P \mid P \vee P \mid \exists x.P$$

$$p ::= P \mid [P] \mid p * p \mid p \vee p \mid \exists x.p$$

$$\frac{s,h \models P}{s,h,H \models P}$$

$$\frac{s,H \models P \qquad h = \varnothing}{s,h,H \models [P]}$$

$$\frac{s,h_1,H \models p_1 \qquad s,h_2,H \models p_2 \qquad h = h_1 \uplus h_2}{s,h,H \models p_1 * p_2}$$



Assertions

$$\frac{s,h \models P}{s,h,H \models P}$$

$$\frac{s,H \models P \qquad h=\varnothing}{s,h,H \models [P]}$$

$$\frac{s,h_1,H \models p_1 \qquad s,h_2,H \models p_2 \qquad h=h_1 \uplus h_2}{s,h,H \models p_1 * p_2}$$

$$[x \mapsto 2] * y \mapsto 3 \;=\; ?$$

$$[x \mapsto 2] * [y \mapsto 3] \;=\; ?$$

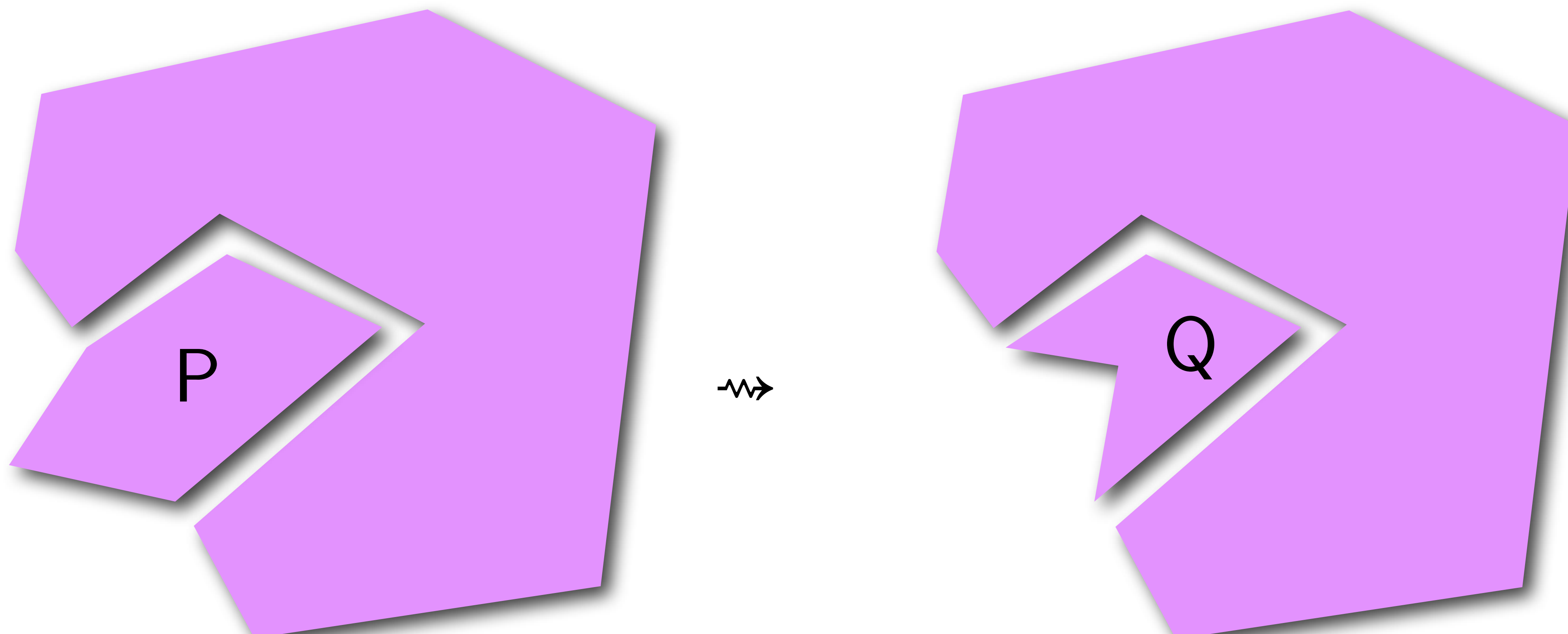
$$[P] * [Q] \;\Leftrightarrow\; [P \wedge Q]$$

Actions

$$\frac{s, H_1 \models P \qquad s, H_2 \models Q}{((s, H_1), (s, H_2)) \models P \rightsquigarrow Q}$$

Actions

$$\frac{s, H_1 \models P \qquad s, H_2 \models Q}{((s, H \uplus H_1), (s, H \uplus H_2)) \models P \rightsquigarrow Q}$$



Proof rules

FRAME

$$\frac{R,G \vdash \{p\}\ C\ \{q\} \quad r \text{ stable under } R \cup G \quad mods(C) \cap fv(r) = \emptyset}{R,G \vdash \{p * r\}\ C\ \{q * r\}}$$

$\forall s,h,H,H'.$
$s,h,H \models p$
$\wedge\ ((s,H),(s,H')) \models R$
$\Rightarrow s,h,H' \models p$

RULE OF CONSEQUENCE

$$\frac{R',G' \vdash \{p'\}\ C\ \{q'\} \quad p \Rightarrow p' \quad q' \Rightarrow q \quad R \subseteq R' \quad G' \subseteq G}{R,G \vdash \{p\}\ C\ \{q\}}$$

BASIC

$$\frac{\vdash \{P\}\ c\ \{Q\}}{R,G \vdash \{P\}\ c\ \{Q\}}$$

Proof rules

SEQUENTIAL COMPOSITION

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

PARALLEL COMPOSITION

$$\frac{R \cup G_2, G_1 \vdash \{p_1\}\ C_1\ \{q_1\} \qquad R \cup G_1, G_2 \vdash \{p_2\}\ C_2\ \{q_2\}}{R,\ G_1 \cup G_2 \vdash \{p_1 * p_2\}\ C_1 \parallel C_2\ \{q_1 * q_2\}}$$

Proof rules

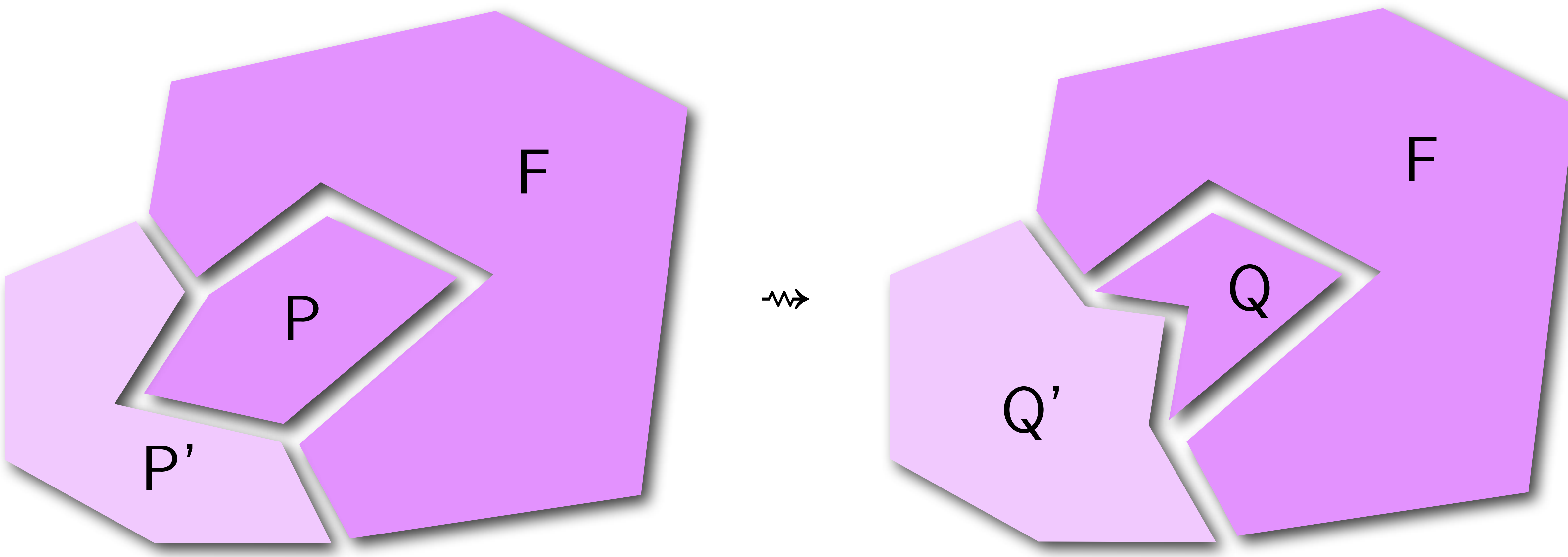
ATOMIC (1)

$$\frac{\bot, G \vdash \{p\}\ \text{atomic}(C)\ \{q\} \qquad p \text{ and } q \text{ stable under } R}{R, G \vdash \{p\}\ \text{atomic}(C)\ \{q\}}$$

ATOMIC (2)

$$\frac{\vdash \{P{*}P'\}\ C\ \{Q{*}Q'\} \qquad P \rightsquigarrow Q \subseteq G \qquad P \text{ and } Q \text{ are precise}}{\bot, G \vdash \{[P{*}F] * P'\}\ \text{atomic}(C)\ \{[Q{*}F] * Q'\}}$$

Proof rules



ATOMIC (2)

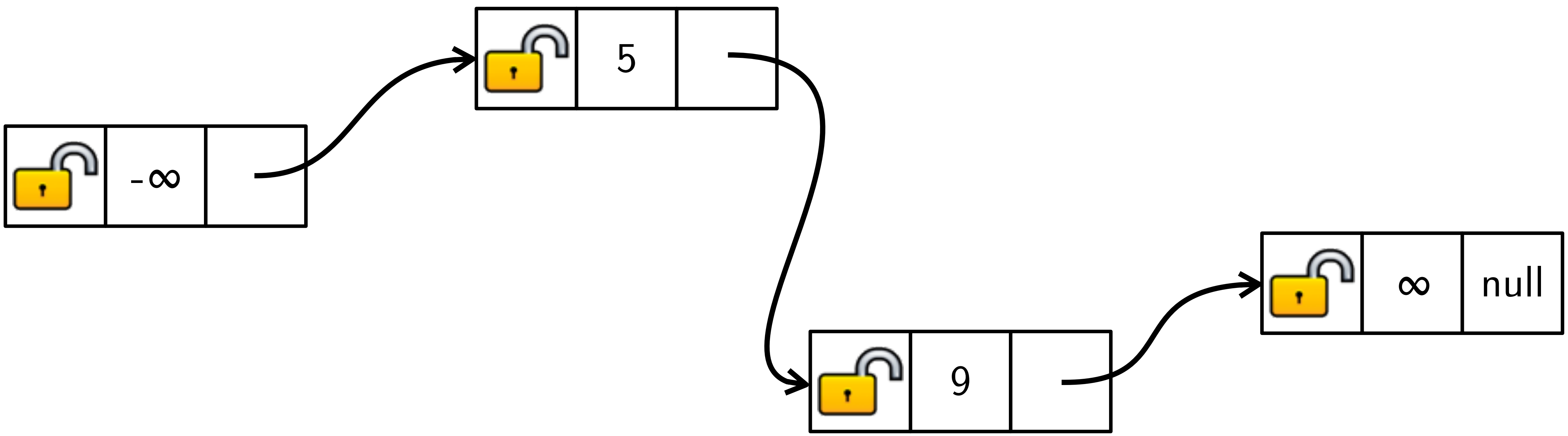
$$\vdash \{P * P'\}\ C\ \{Q * Q'\}$$

$$\frac{P \rightsquigarrow Q \subseteq G \qquad\qquad P \text{ and } Q \text{ are precise}}{\bot, G \vdash \{[P * F] * P'\}\ \text{atomic}(C)\ \{[Q * F] * Q'\}}$$

Lecture plan

1. Introducing RGSep
2. Verification of a fine-grained concurrent datastructure
3. RG and CSL as special cases
4. Extensions, limitations and further work

Lock-coupling list



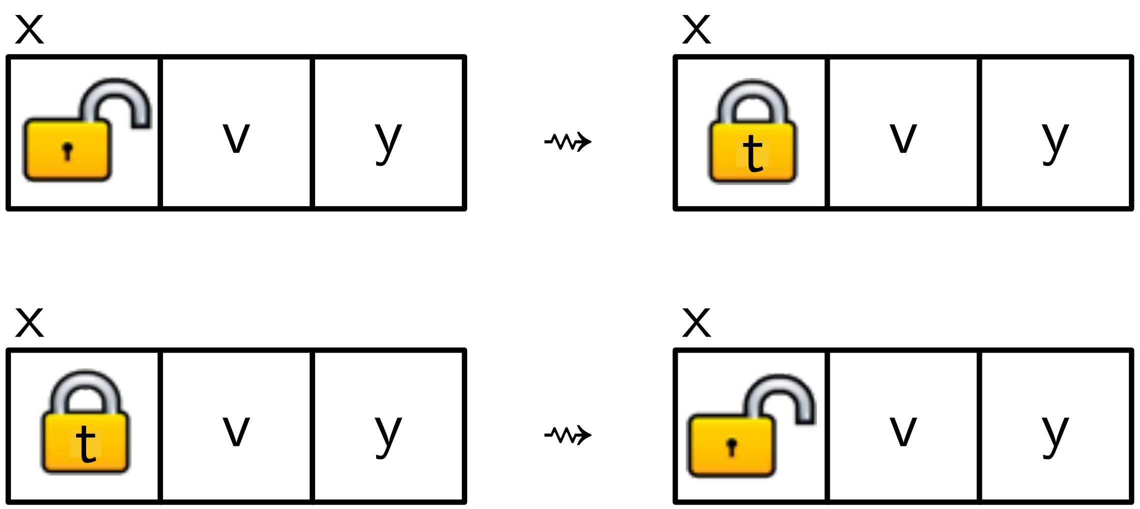
$$\text{lock(p)} \overset{\text{def}}{=} \textbf{atomic}_{\text{p.lock} = 0} \text{ (p.lock := tid)}$$

$$\text{unlock(p)} \overset{\text{def}}{=} \textbf{atomic} \text{ (p.lock := 0)}$$

Lock-coupling list

LOCK_t :

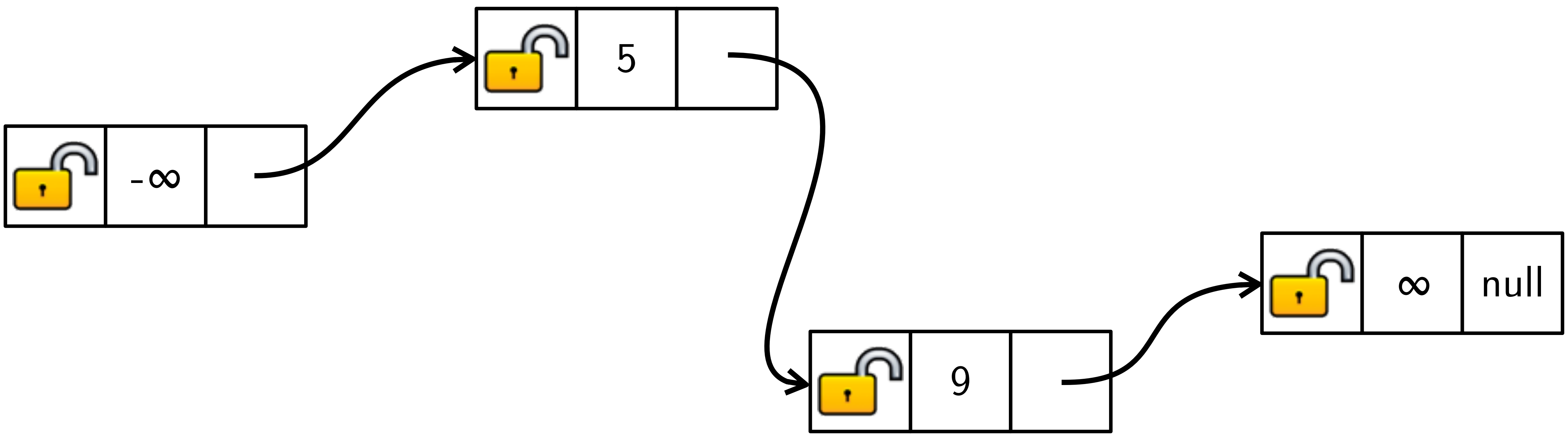
UNLOCK_t :



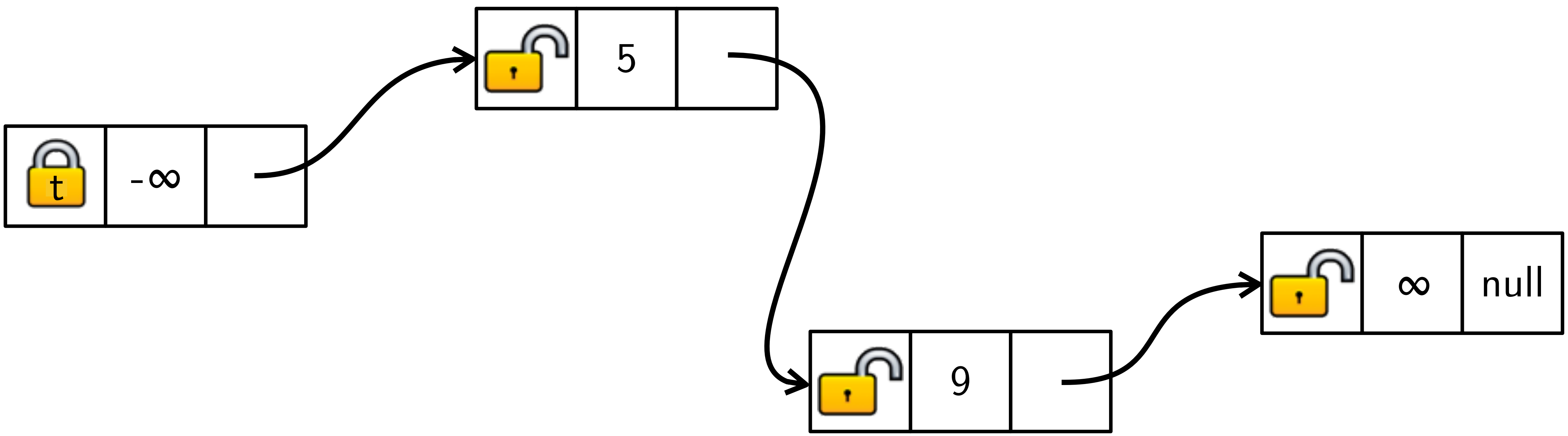
$$\mathrm{lock(p)} \stackrel{\text{def}}{=} \textbf{atomic}_{\mathrm{p.lock}\,=\,0}\ (\ \mathrm{p.lock} := \mathrm{tid}\ )$$

$$\mathrm{unlock(p)} \stackrel{\text{def}}{=} \textbf{atomic}\ (\ \mathrm{p.lock} := 0\ )$$

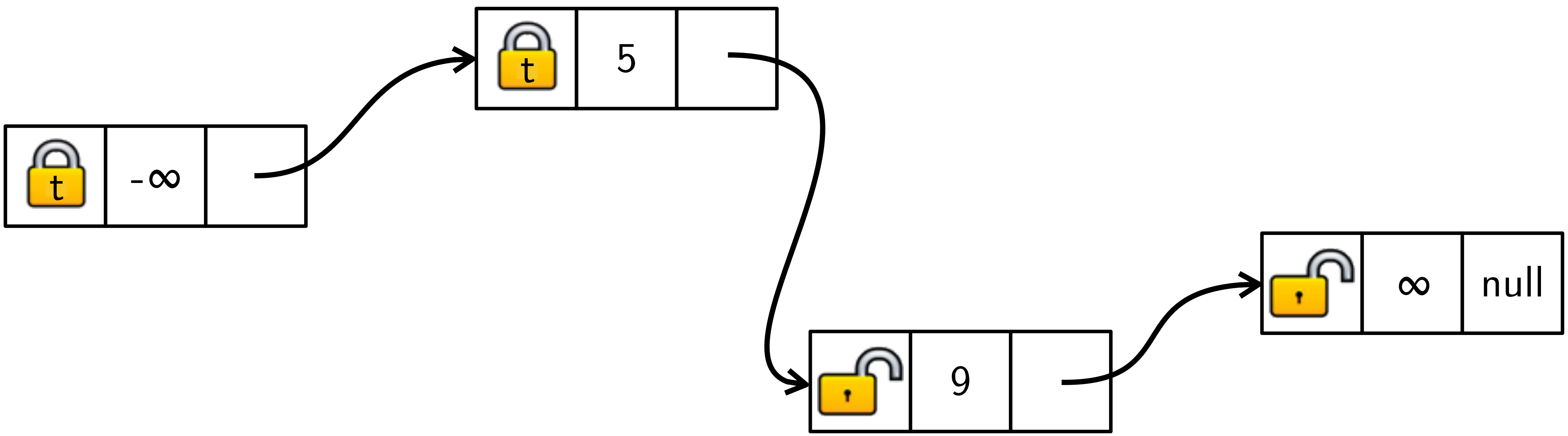
Lock-coupling list



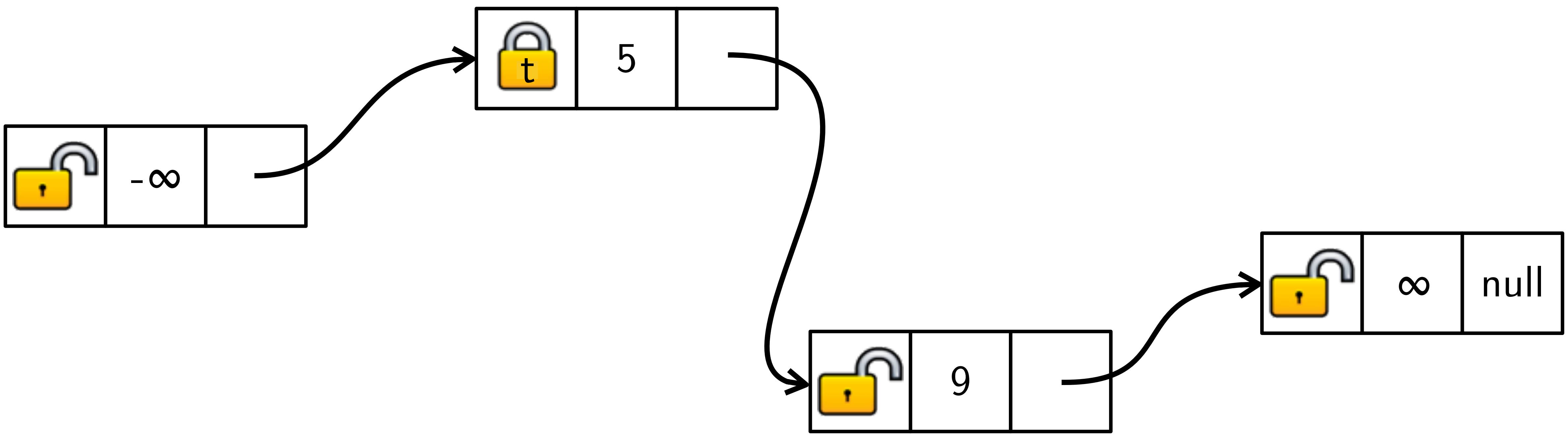
Lock-coupling list



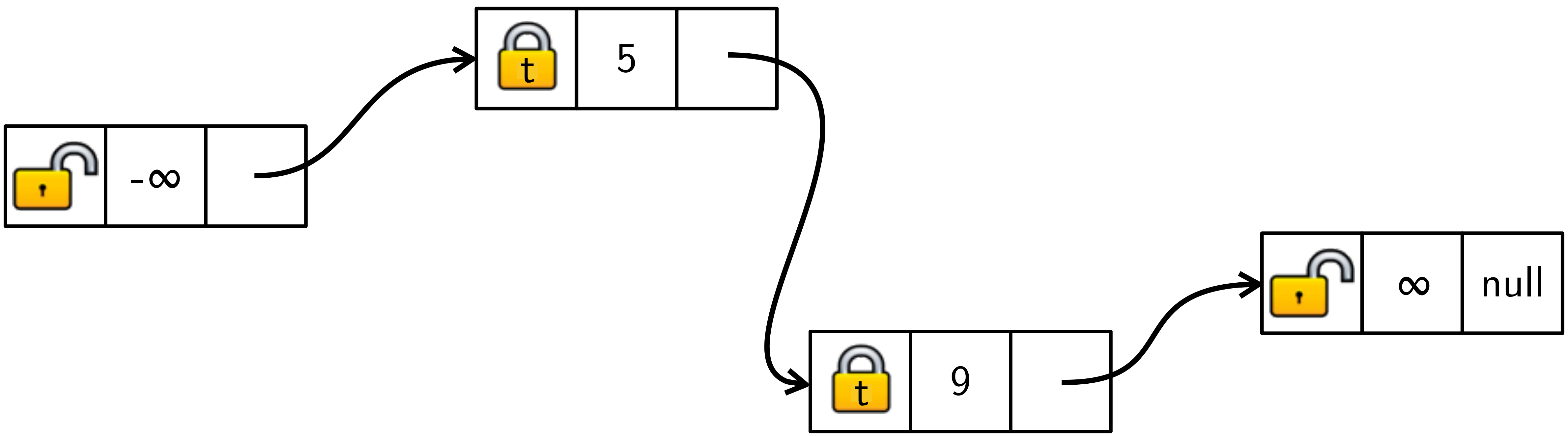
Lock-coupling list



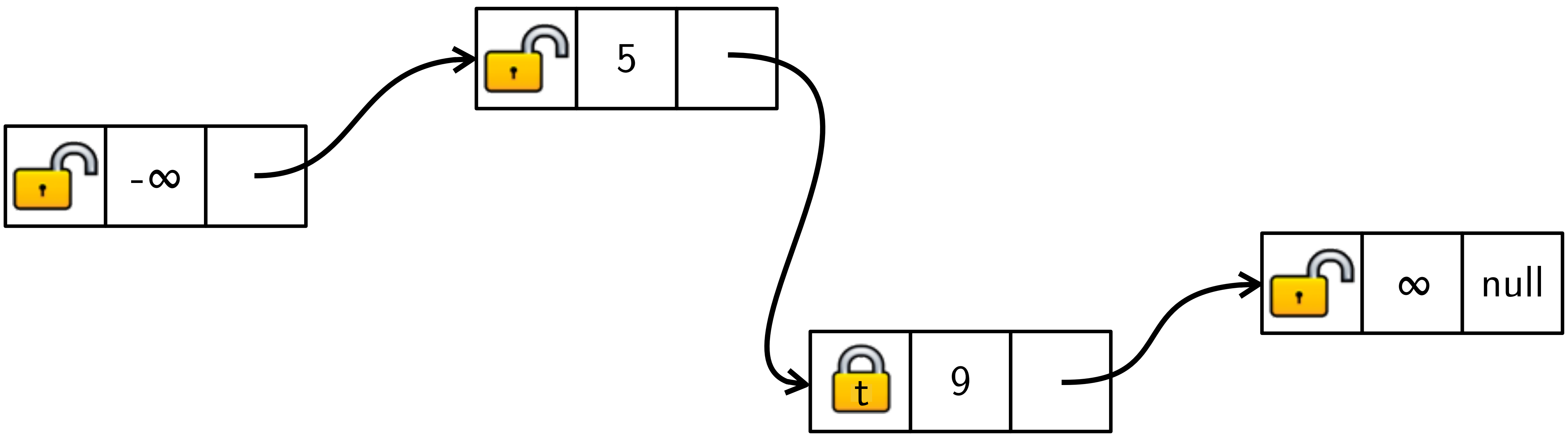
Lock-coupling list



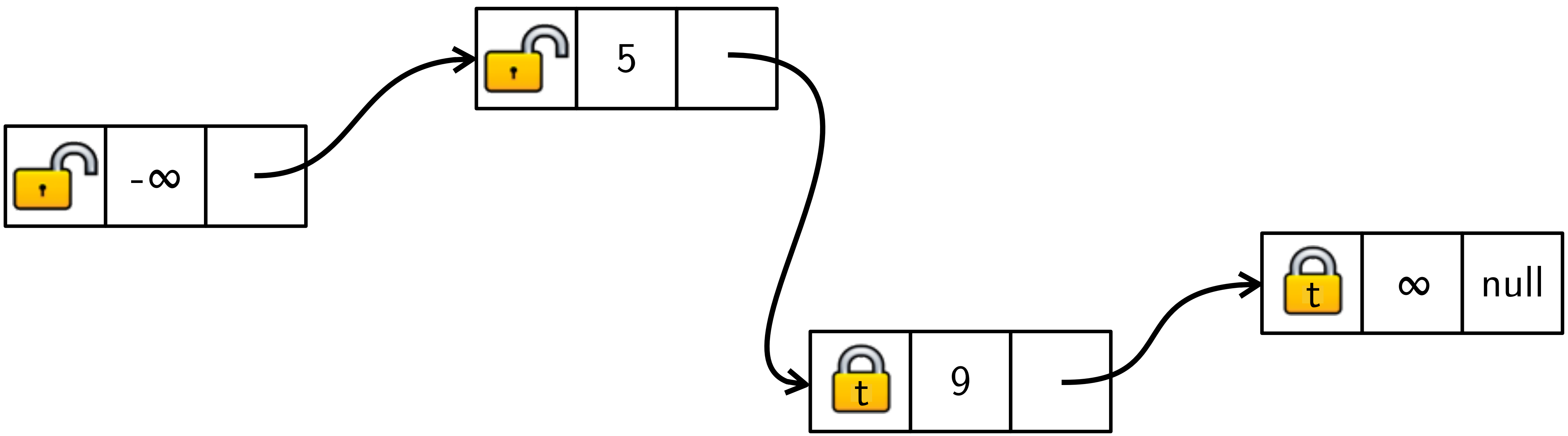
Lock-coupling list



Lock-coupling list



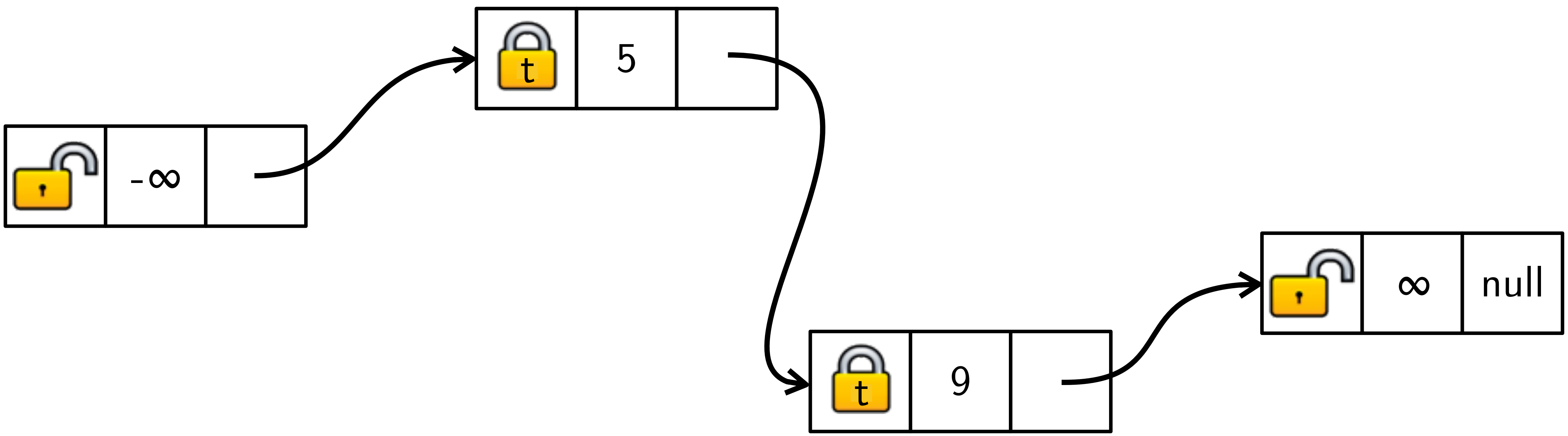
Lock-coupling list



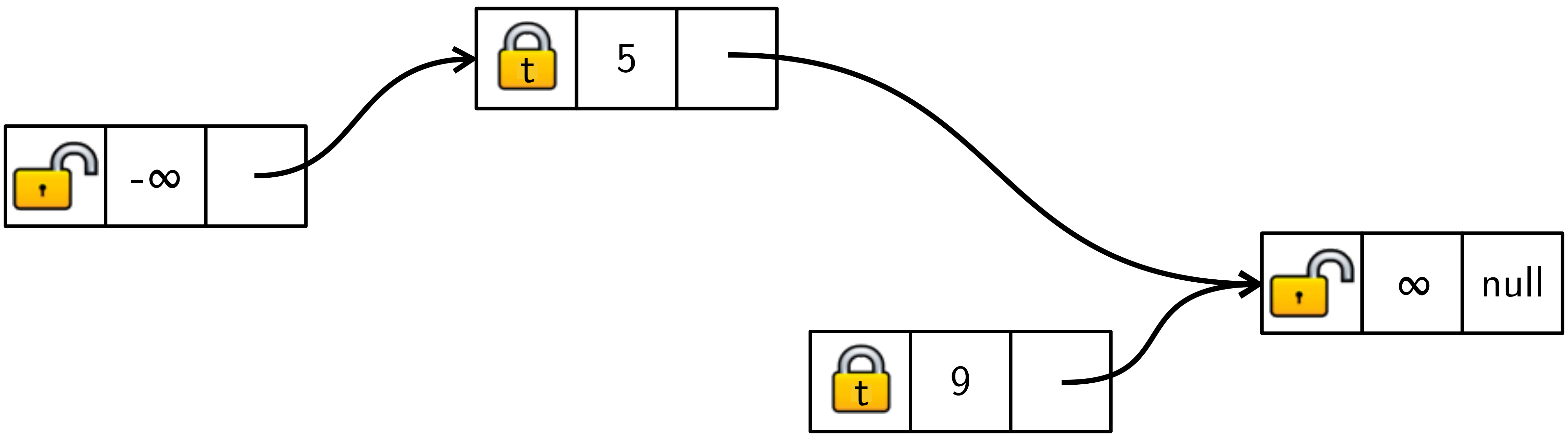
Lock-coupling list

```
remove(e) ≝
  local p,c,n,v; p := Head; lock(p); ⟨c := p.next⟩; ⟨v := c.value⟩;
  while (v < e) (
    lock(c); unlock(p); p := c; ⟨c := p.next⟩; ⟨v := c.value⟩;
  )
  if (v = e) (
    lock(c); ⟨n := c.next⟩; ⟨p.next := n⟩; unlock(p); dispose(c);
  ) else unlock(p)
```

Lock-coupling list



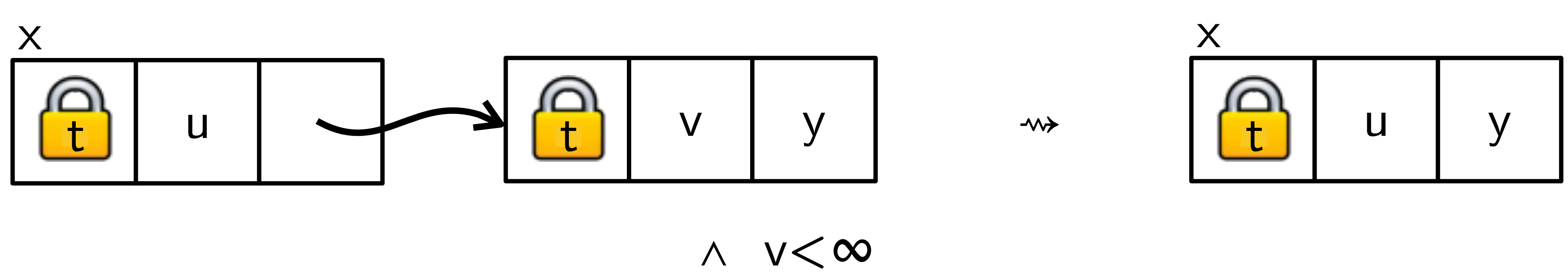
Lock-coupling list



Lock-coupling list

```
remove(e) ≝
  local p,c,n,v; p := Head; lock(p); ⟨c := p.next⟩; ⟨v := c.value⟩;
  while (v < e) (
    lock(c); unlock(p); p := c; ⟨c := p.next⟩; ⟨v := c.value⟩;
  )
  if (v = e) (
    lock(c); ⟨n := c.next⟩; ⟨p.next := n⟩; unlock(p); dispose(c);
  ) else unlock(p)
```

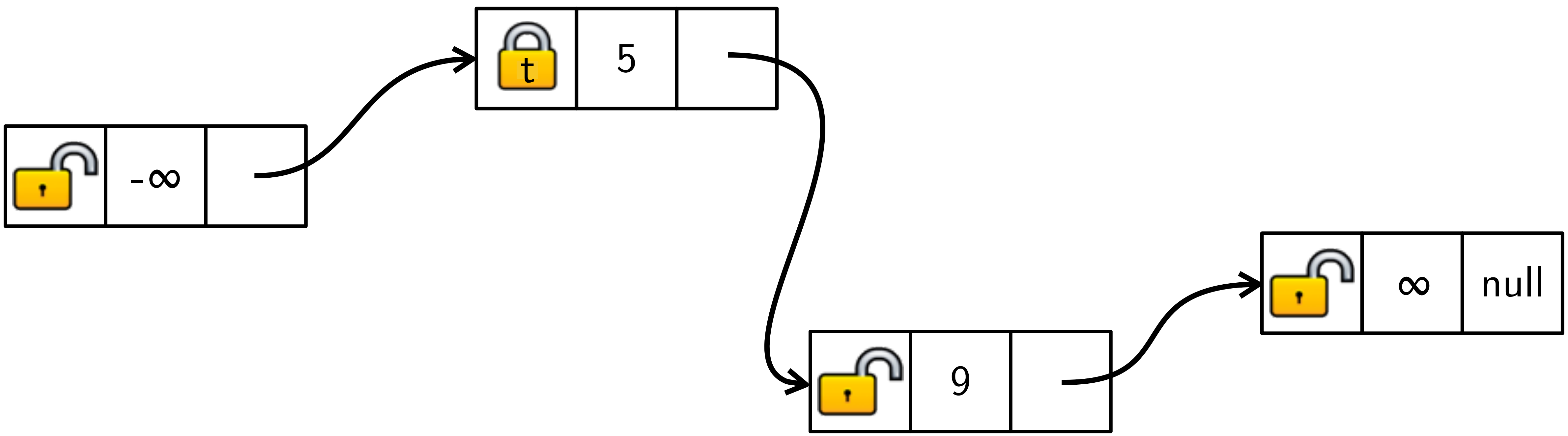
$REMOVE_t$:



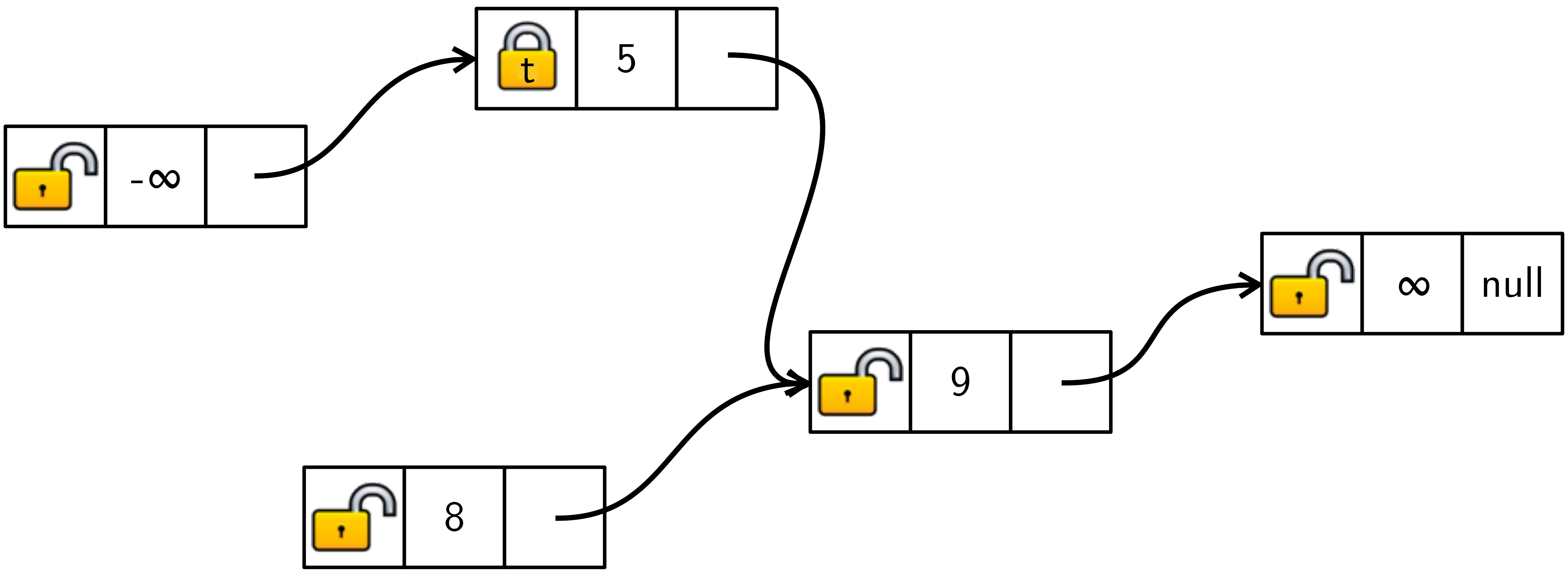
Lock-coupling list

```
insert(e) ≝
  local p,c,n,v; p := Head; lock(p); ⟨c := p.next⟩; ⟨v := c.value⟩;
  while (v < e) (
    lock(c); unlock(p); p := c; ⟨c := p.next⟩; ⟨v := c.value⟩;
  )
  if (v ≠ e) (
    n := cons(0, e, c); ⟨p.next := n⟩;
  )
  unlock(p)
```

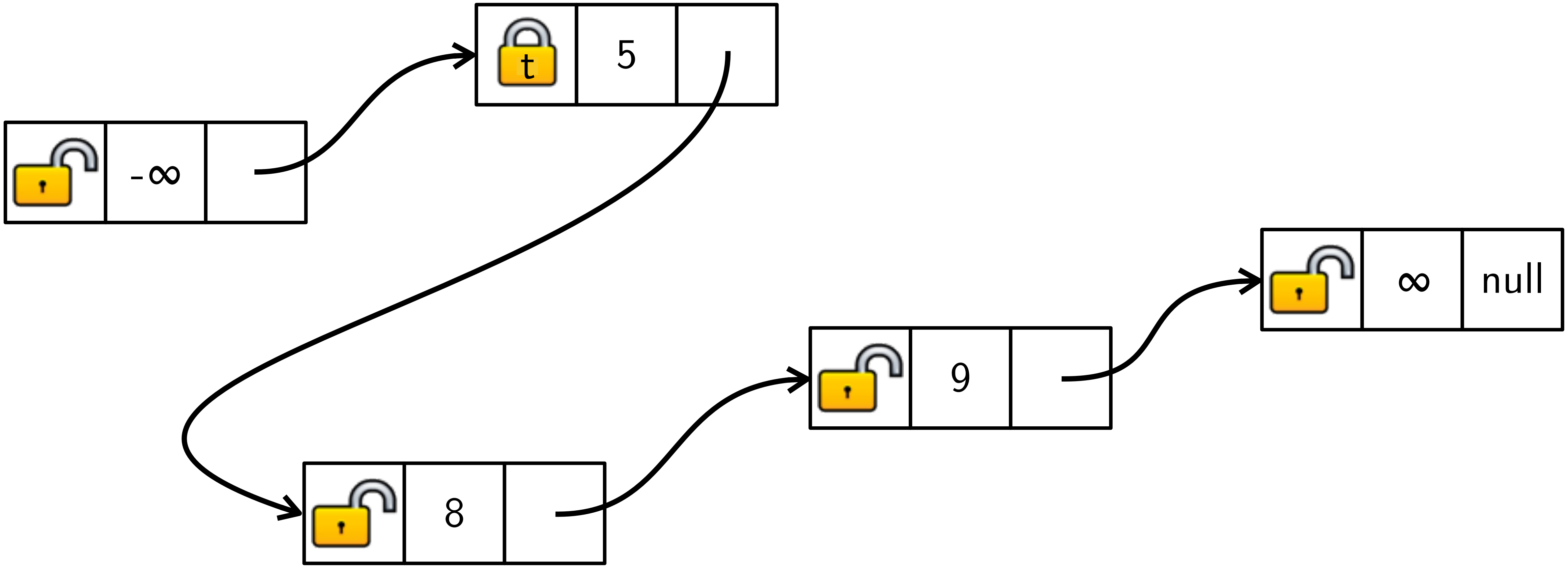
Lock-coupling list



Lock-coupling list

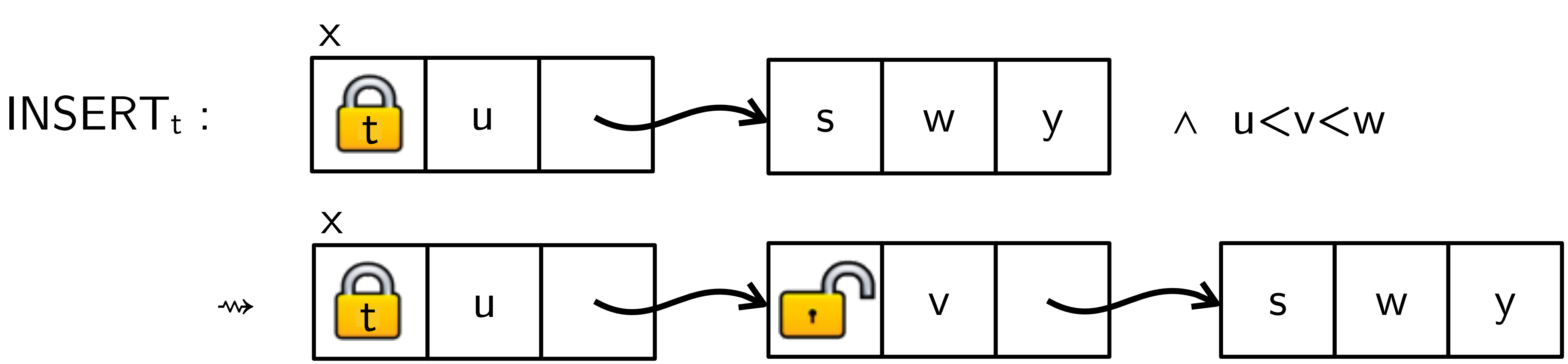


Lock-coupling list

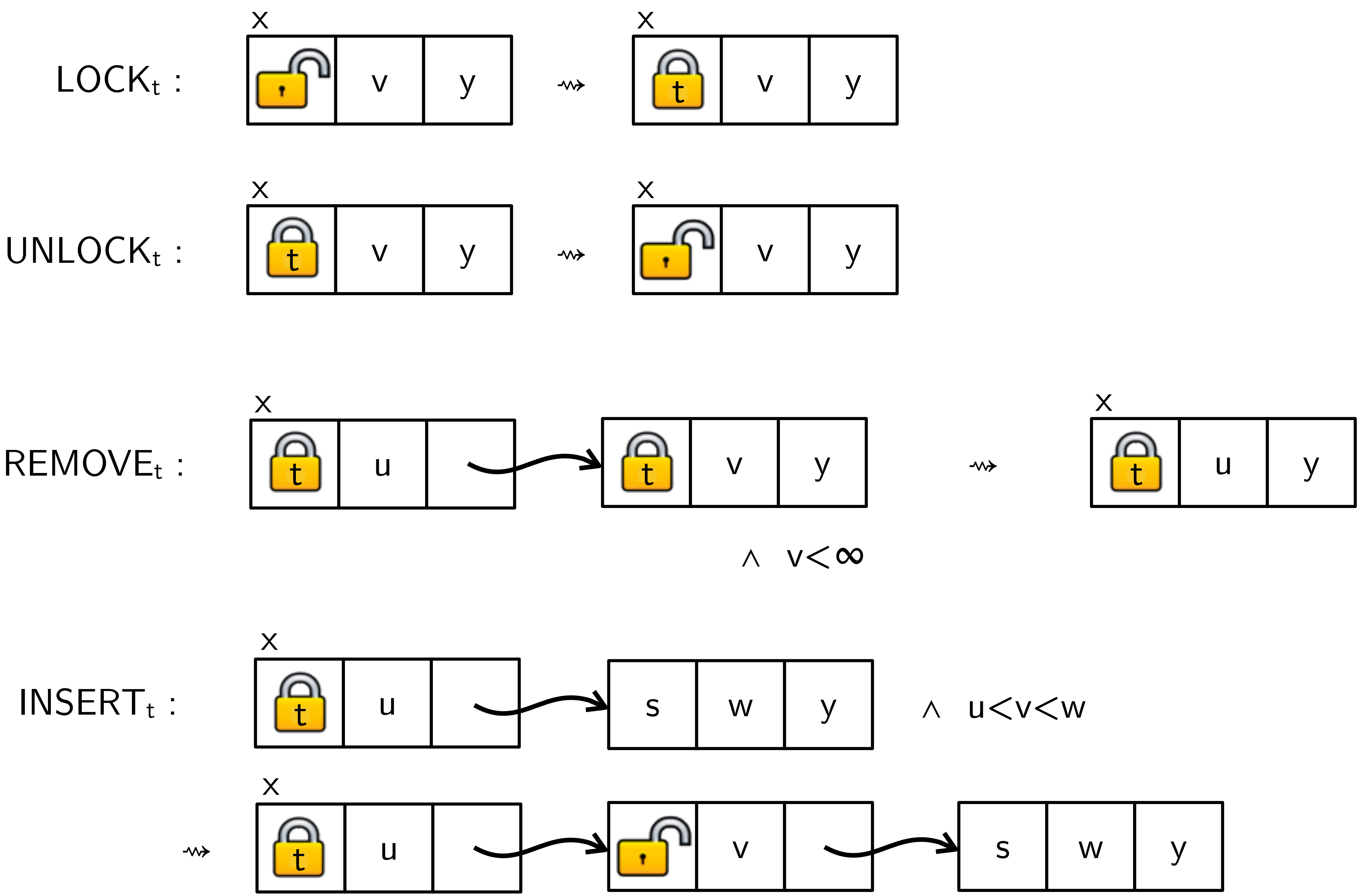


Lock-coupling list

insert(e) $\stackrel{\text{def}}{=}$
local p,c,n,v; p := Head; lock(p); ⟨c := p.next⟩; ⟨v := c.value⟩;
while (v < e) (
lock(c); unlock(p); p := c; ⟨c := p.next⟩; ⟨v := c.value⟩;
)
if (v ≠ e) (
n := **cons**(0, e, c); ⟨p.next := n⟩;
)
unlock(p)



Lock-coupling list



Lock-coupling list

For a thread with thread-id t, set:

$R \;=\; \bigcup_{t' \neq t} \bigcup\{\, \text{LOCK}_{t'},\ \text{UNLOCK}_{t'},\ \text{REMOVE}_{t'},\ \text{INSERT}_{t'} \,\}$

$G \;=\; \bigcup\{\, \text{LOCK}_{t},\ \text{UNLOCK}_{t},\ \text{REMOVE}_{t},\ \text{INSERT}_{t} \,\}$

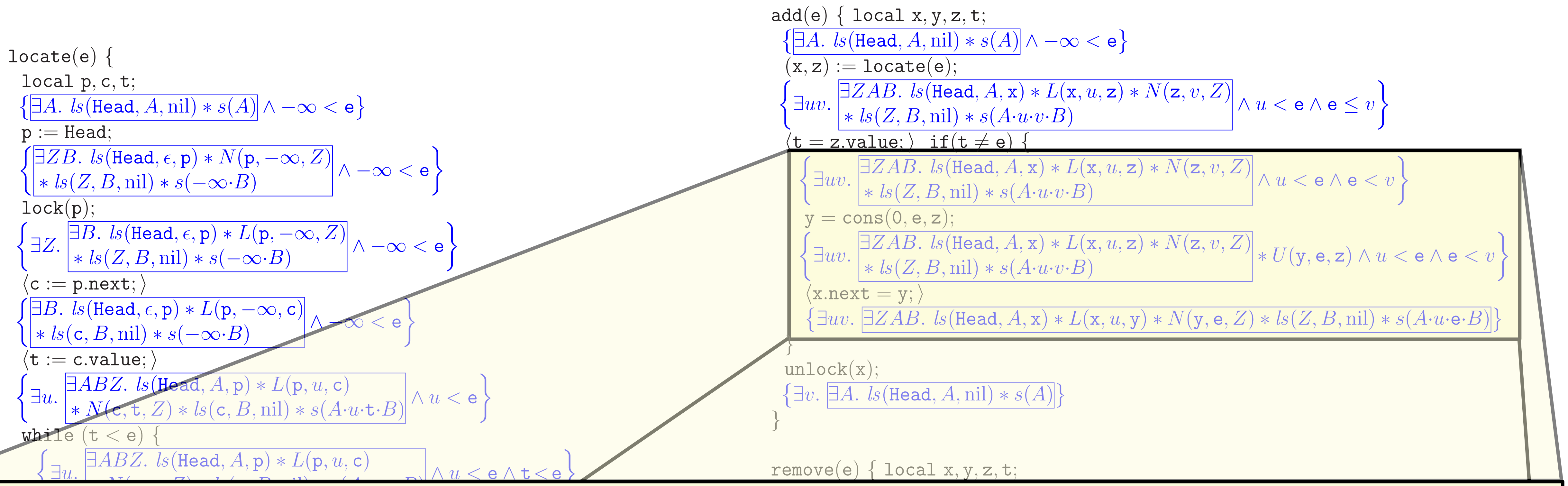
Lock-coupling list

```
locate(e) {
  local p, c, t;
  {[∃A. ls(Head, A, nil) ∗ s(A)] ∧ −∞ < e}
  p := Head;
  {[∃ZB. ls(Head, ε, p) ∗ N(p, −∞, Z)
    ∗ ls(Z, B, nil) ∗ s(−∞·B)] ∧ −∞ < e}
  lock(p);
  {∃Z. [∃B. ls(Head, ε, p) ∗ L(p, −∞, Z)
    ∗ ls(Z, B, nil) ∗ s(−∞·B)] ∧ −∞ < e}
  ⟨c := p.next;⟩
  {[∃B. ls(Head, ε, p) ∗ L(p, −∞, c)
    ∗ ls(c, B, nil) ∗ s(−∞·B)] ∧ −∞ < e}
  ⟨t := c.value;⟩
  {∃u. [∃ABZ. ls(Head, A, p) ∗ L(p, u, c)
    ∗ N(c, t, Z) ∗ ls(c, B, nil) ∗ s(A·u·t·B)] ∧ u < e}
  while (t < e) {
    {∃u. [∃ABZ. ls(Head, A, p) ∗ L(p, u, c)
      ∗ N(c, t, Z) ∗ ls(c, B, nil) ∗ s(A·u·t·B)] ∧ u < e ∧ t<e}
    lock(c);
    {∃uZ. [∃AB. ls(Head, A, p) ∗ L(p, u, c)
      ∗ L(c, t, Z) ∗ ls(Z, B, nil) ∗ s(A·u·t·B)] ∧ t < e}
    unlock(p);
    {∃Z. [∃AB. ls(Head, A, c) ∗ L(c, t, Z)
      ∗ ls(Z, B, nil) ∗ s(A·t·B)] ∧ t < e}
    p := c;
    {∃uZ. [∃AB. ls(Head, A, p) ∗ L(p, u, Z)
      ∗ ls(Z, B, nil) ∗ s(A·u·B)] ∧ u < e}
    ⟨c := p.next;⟩
    {∃u. [∃AB. ls(Head, A, p) ∗ L(p, u, c)
      ∗ ls(c, B, nil) ∗ s(A·u·B)] ∧ u < e}
    ⟨t := c.value;⟩
    {∃u. [∃ABZ. ls(Head, A, p) ∗ L(p, u, c)
      ∗ N(c, t, Z) ∗ ls(Z, B, nil) ∗ s(A·u·t·B)] ∧ u < e}
  }
  {∃uv. [∃ABZ. ls(Head, A, p) ∗ L(p, u, c)
    ∗ N(c, v, Z) ∗ ls(Z, B, nil) ∗ s(A·u·v·B)] ∧ u < e ∧ e ≤ v}
  return (p, c);
}
```

```
add(e) { local x, y, z, t;
  {[∃A. ls(Head, A, nil) ∗ s(A)] ∧ −∞ < e}
  (x, z) := locate(e);
  {∃uv. [∃ZAB. ls(Head, A, x) ∗ L(x, u, z) ∗ N(z, v, Z)
    ∗ ls(Z, B, nil) ∗ s(A·u·v·B)] ∧ u < e ∧ e ≤ v}
  ⟨t = z.value;⟩ if(t ≠ e) {
    {∃uv. [∃ZAB. ls(Head, A, x) ∗ L(x, u, z) ∗ N(z, v, Z)
      ∗ ls(Z, B, nil) ∗ s(A·u·v·B)] ∧ u < e ∧ e < v}
    y = cons(0, e, z);
    {∃uv. [∃ZAB. ls(Head, A, x) ∗ L(x, u, z) ∗ N(z, v, Z)
      ∗ ls(Z, B, nil) ∗ s(A·u·v·B)] ∗ U(y, e, z) ∧ u < e ∧ e < v}
    ⟨x.next = y;⟩
    {∃uv. [∃ZAB. ls(Head, A, x) ∗ L(x, u, y) ∗ N(y, e, Z) ∗ ls(Z, B, nil) ∗ s(A·u·e·B)]}
  }
  unlock(x);
  {∃v. [∃A. ls(Head, A, nil) ∗ s(A)]}
}
```

```
remove(e) { local x, y, z, t;
  {[∃A. ls(Head, A, nil) ∗ s(A)] ∧ −∞ < e ∧ e < +∞}
  (x, y) = locate(e);
  {∃uv. [∃ZAB. ls(Head, A, x) ∗ L(x, u, y) ∗ N(y, v, Z)
    ∗ ls(Z, B, nil) ∗ s(A·u·v·B)] ∧ u < e ∧ e ≤ v ∧ e < +∞}
  ⟨t = y.value;⟩ if(t = e) {
    {∃u. [∃ZAB. ls(Head, A, x) ∗ L(x, u, y) ∗ N(y, e, Z)
      ∗ ls(Z, B, nil) ∗ s(A·u·e·B)] ∧ e < +∞}
    lock(y);
    {∃u. [∃ZAB. ls(Head, A, x) ∗ L(x, u, y) ∗ L(y, e, Z)
      ∗ ls(Z, B, nil) ∗ s(A·u·e·B)] ∧ e < +∞}
    ⟨z := y.next;⟩
    {∃u. [∃AB. ls(Head, A, x) ∗ L(x, u, y) ∗ L(y, e, z)
      ∗ ls(z, B, nil) ∗ s(A·u·e·B)] ∧ e < +∞}
    ⟨x.next := z;⟩
    {∃u. [∃AB. ls(Head, A, x) ∗ L(x, u, z) ∗ ls(z, B, nil) ∗ s(A·u·B)] ∗ L(y, e, z)}
    unlock(x);
    {[∃A. ls(Head, A, nil) ∗ s(A)] ∗ L(y, e, z)}
    dispose(y);
  } else { {∃u. [∃ZAB. ls(Head, A, x) ∗ L(x, u, y) ∗ ls(y, B, nil) ∗ s(A·u·B)]}
    unlock(x); }
  {[∃A. ls(Head, A, nil) ∗ s(A)]}
}
```

Lock-coupling list



$$\left\{\exists uv.\ \boxed{\begin{array}{l}\exists ZAB.\ ls(\mathtt{Head}, A, \mathtt{x}) * L(\mathtt{x}, u, \mathtt{z}) * N(\mathtt{z}, v, Z)\\ * \ ls(Z, B, \mathrm{nil}) * s(A{\cdot}u{\cdot}v{\cdot}B)\end{array}} \wedge u < \mathtt{e} \wedge \mathtt{e} < v\right\}$$

$\mathtt{y} = \mathtt{cons}(0, \mathtt{e}, \mathtt{z});$

$$\left\{\exists uv.\ \boxed{\begin{array}{l}\exists ZAB.\ ls(\mathtt{Head}, A, \mathtt{x}) * L(\mathtt{x}, u, \mathtt{z}) * N(\mathtt{z}, v, Z)\\ * \ ls(Z, B, \mathrm{nil}) * s(A{\cdot}u{\cdot}v{\cdot}B)\end{array}} * U(\mathtt{y}, \mathtt{e}, \mathtt{z}) \wedge u < \mathtt{e} \wedge \mathtt{e} < v\right\}$$

$\langle \mathtt{x.next} = \mathtt{y}; \rangle$

$$\left\{\exists uv.\ \boxed{\exists ZAB.\ ls(\mathtt{Head}, A, \mathtt{x}) * L(\mathtt{x}, u, \mathtt{y}) * N(\mathtt{y}, \mathtt{e}, Z) * ls(Z, B, \mathrm{nil}) * s(A{\cdot}u{\cdot}\mathtt{e}{\cdot}B)}\right\}$$

Lock-coupling list

locate(e) {
local p, c, t;

$\{\boxed{\exists A.\ ls(\texttt{Head}, A, \mathrm{nil}) * s(A)} \wedge -\infty < \mathtt{e}\}$

p := Head;

$\left\{\boxed{\begin{array}{l}\exists ZB.\ ls(\texttt{Head}, \epsilon, \mathtt{p}) * N(\mathtt{p}, -\infty, Z) \\ * ls(Z, B, \mathrm{nil}) * s(-\infty \cdot B)\end{array}} \wedge -\infty < \mathtt{e}\right\}$

lock(p);

$\left\{\exists Z.\ \boxed{\begin{array}{l}\exists B.\ ls(\texttt{Head}, \epsilon, \mathtt{p}) * L(\mathtt{p}, -\infty, Z) \\ * ls(Z, B, \mathrm{nil}) * s(-\infty \cdot B)\end{array}} \wedge -\infty < \mathtt{e}\right\}$

[…]

unlock(p);

$\left\{\exists Z.\ \boxed{\begin{array}{l}\exists AB.\ ls(\texttt{Head}, A, \mathtt{c}) * L(\mathtt{c}, \mathtt{t}, Z) \\ * ls(Z, B, \mathrm{nil}) * s(A \cdot \mathtt{t} \cdot B)\end{array}} \wedge \mathtt{t} < \mathtt{e}\right\}$

p := c;

$\left\{\exists uZ.\ \boxed{\begin{array}{l}\exists AB.\ ls(\texttt{Head}, A, \mathtt{p}) * L(\mathtt{p}, u, Z) \\ * ls(Z, B, \mathrm{nil}) * s(A \cdot u \cdot B)\end{array}} \wedge u < \mathtt{e}\right\}$

⟨c := p.next;⟩

$\left\{\exists u.\ \boxed{\begin{array}{l}\exists AB.\ ls(\texttt{Head}, A, \mathtt{p}) * L(\mathtt{p}, u, \mathtt{c}) \\ * ls(\mathtt{c}, B, \mathrm{nil}) * s(A \cdot u \cdot B)\end{array}} \wedge u < \mathtt{e}\right\}$

⟨t := c.value;⟩

$\left\{\exists u.\ \boxed{\begin{array}{l}\exists ABZ.\ ls(\texttt{Head}, A, \mathtt{p}) * L(\mathtt{p}, u, \mathtt{c}) \\ * N(\mathtt{c}, \mathtt{t}, Z) * ls(Z, B, \mathrm{nil}) * s(A \cdot u \cdot \mathtt{t} \cdot B)\end{array}} \wedge u < \mathtt{e}\right\}$

}

$\left\{\exists uv.\ \boxed{\begin{array}{l}\exists ABZ.\ ls(\texttt{Head}, A, \mathtt{p}) * L(\mathtt{p}, u, \mathtt{c}) \\ * N(\mathtt{c}, v, Z) * ls(Z, B, \mathrm{nil}) * s(A \cdot u \cdot v \cdot B)\end{array}} \wedge u < \mathtt{e} \wedge \mathtt{e} \leq v\right\}$

return (p, c);

}

add(e) { local x, y, z, t;

$\{\boxed{\exists A.\ ls(\texttt{Head}, A, \mathrm{nil}) * s(A)} \wedge -\infty < \mathtt{e}\}$

(x, z) := locate(e);

$\left\{\exists uv.\ \boxed{\begin{array}{l}\exists ZAB.\ ls(\texttt{Head}, A, \mathtt{x}) * L(\mathtt{x}, u, \mathtt{z}) * N(\mathtt{z}, v, Z) \\ * ls(Z, B, \mathrm{nil}) * s(A \cdot u \cdot v \cdot B)\end{array}} \wedge u < \mathtt{e} \wedge \mathtt{e} \leq v\right\}$

⟨t = z.value;⟩ if(t ≠ e) {

$\left\{\exists uv.\ \boxed{\begin{array}{l}\exists ZAB.\ ls(\texttt{Head}, A, \mathtt{x}) * L(\mathtt{x}, u, \mathtt{z}) * N(\mathtt{z}, v, Z) \\ * ls(Z, B, \mathrm{nil}) * s(A \cdot u \cdot v \cdot B)\end{array}} \wedge u < \mathtt{e} \wedge \mathtt{e} < v\right\}$

y = cons(0, e, z);

$\left\{\exists uv.\ \boxed{\begin{array}{l}\exists ZAB.\ ls(\texttt{Head}, A, \mathtt{x}) * L(\mathtt{x}, u, \mathtt{z}) * N(\mathtt{z}, v, Z) \\ \ldots\end{array}} * U(\mathtt{y}, \mathtt{e}, \mathtt{z}) \wedge u < \mathtt{e} \wedge \mathtt{e} < v\right\}$

[…]

⟨t = y.value;⟩ if(t = e) {

$\left\{\exists u.\ \boxed{\begin{array}{l}\exists ZAB.\ ls(\texttt{Head}, A, \mathtt{x}) * L(\mathtt{x}, u, \mathtt{y}) * N(\mathtt{y}, \mathtt{e}, Z) \\ * ls(Z, B, \mathrm{nil}) * s(A \cdot u \cdot \mathtt{e} \cdot B)\end{array}} \wedge \mathtt{e} < +\infty\right\}$

lock(y);

$\left\{\exists u.\ \boxed{\begin{array}{l}\exists ZAB.\ ls(\texttt{Head}, A, \mathtt{x}) * L(\mathtt{x}, u, \mathtt{y}) * L(\mathtt{y}, \mathtt{e}, Z) \\ * ls(Z, B, \mathrm{nil}) * s(A \cdot u \cdot \mathtt{e} \cdot B)\end{array}} \wedge \mathtt{e} < +\infty\right\}$

⟨z := y.next;⟩

$\left\{\exists u.\ \boxed{\begin{array}{l}\exists AB.\ ls(\texttt{Head}, A, \mathtt{x}) * L(\mathtt{x}, u, \mathtt{y}) * L(\mathtt{y}, \mathtt{e}, \mathtt{z}) \\ * ls(\mathtt{z}, B, \mathrm{nil}) * s(A \cdot u \cdot \mathtt{e} \cdot B)\end{array}} \wedge \mathtt{e} < +\infty\right\}$

⟨x.next := z;⟩

$\{\exists u.\ \boxed{\exists AB.\ ls(\texttt{Head}, A, \mathtt{x}) * L(\mathtt{x}, u, \mathtt{z}) * ls(\mathtt{z}, B, \mathrm{nil}) * s(A \cdot u \cdot B)} * L(\mathtt{y}, \mathtt{e}, \mathtt{z})\}$

unlock(x);

$\{\boxed{\exists A.\ ls(\texttt{Head}, A, \mathrm{nil}) * s(A)} * L(\mathtt{y}, \mathtt{e}, \mathtt{z})\}$

dispose(y);

} else { $\{\exists u.\ \boxed{\exists ZAB.\ ls(\texttt{Head}, A, \mathtt{x}) * L(\mathtt{x}, u, \mathtt{y}) * ls(\mathtt{y}, B, \mathrm{nil}) * s(A \cdot u \cdot B)}\}$

unlock(x); }

$\{\boxed{\exists A.\ ls(\texttt{Head}, A, \mathrm{nil}) * s(A)}\}$

}

$$\left\{\exists u.\ \boxed{\begin{array}{l}\exists AB.\ ls(\texttt{Head}, A, \mathtt{x}) * L(\mathtt{x}, u, \mathtt{y}) * L(\mathtt{y}, \mathtt{e}, \mathtt{z}) \\ * ls(\mathtt{z}, B, \mathrm{nil}) * s(A \cdot u \cdot \mathtt{e} \cdot B)\end{array}} \wedge \mathtt{e} < +\infty\right\}$$

⟨x.next := z;⟩

$$\{\exists u.\ \boxed{\exists AB.\ ls(\texttt{Head}, A, \mathtt{x}) * L(\mathtt{x}, u, \mathtt{z}) * ls(\mathtt{z}, B, \mathrm{nil}) * s(A \cdot u \cdot B)} * L(\mathtt{y}, \mathtt{e}, \mathtt{z})\}$$

Lecture plan

1. Introducing RGSep
2. Verification of a fine-grained concurrent datastructure
3. RG and CSL as special cases
4. Extensions, limitations and further work

Special cases

PARALLEL COMPOSITION

$$\frac{R \cup G_2, G_1 \vdash \{p_1\}\ C_1\ \{q_1\} \qquad R \cup G_1, G_2 \vdash \{p_2\}\ C_2\ \{q_2\}}{R,\ G_1 \cup G_2 \vdash \{p_1 * p_2\}\ C_1 \parallel C_2\ \{q_1 * q_2\}}$$

$$R,G \vdash_{RG} \{P\}\ C\ \{Q\} \quad \longleftrightarrow \quad R,G \vdash_{RGSep} \{\ [P]\ \}\ C\ \{\ [Q]\ \}$$

$$\vdash_{SL} \{P\}\ C\ \{Q\} \quad \longleftrightarrow \quad \bot,\bot \vdash_{RGSep} \{P\}\ C\ \{Q\}$$

$$J \vdash_{CSL} \{P\}\ C\ \{Q\} \quad \longleftrightarrow \quad J \rightsquigarrow J,\ J \rightsquigarrow J \vdash_{RGSep} \{\ P * [J]\ \}\ C\ \{\ Q * [J]\ \}$$

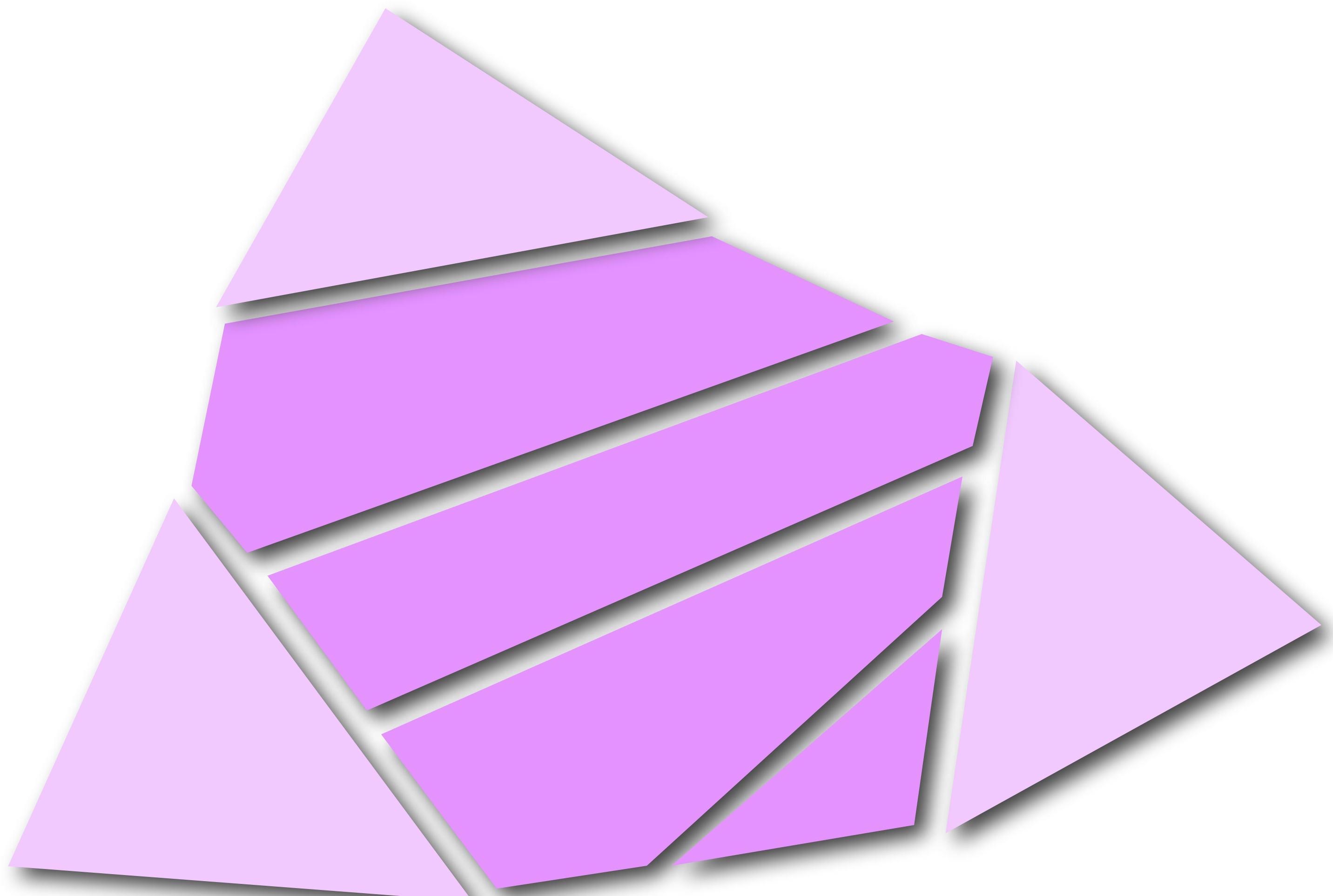
Lecture plan

1. Introducing RGSep
2. Verification of a fine-grained concurrent datastructure
3. RG and CSL as special cases
4. Extensions, limitations and further work

Multiple regions

State = { (s,h,H) ∈ Store × Heap × Heap | $h \perp H$ }

State = { (s,h,H) ∈ Store × Heap × (RName ⇀ Heap) | $h \perp H(r_1) \perp \ldots \perp H(r_n)$ }



Local Rely-Guarantee

i := 0; j := 1; [x] := |A|; [y] := |A|;

```
while i<min([x],[y]) do          ||  while j<min([x],[y]) do
  if A[i]>0 then                 ||    if A[j]>0 then
    [x]:=i                       ||      [y]:=j
  else                           ||    else
    i:=i+2                       ||      j:=j+2
  end if                         ||    end if
end while                        ||  end while
```

r := min([x],[y])

References

(1) Viktor Vafeiadis. *Modular fine-grained concurrency verification.* PhD thesis, University of Cambridge, 2007. Available from: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-726.html

(2) Viktor Vafeiadis and Matthew Parkinson. *A marriage of Rely/Guarantee and Separation Logic.* CONCUR, 2007. Available from: http://www.mpi-sws.org/~viktor/papers/concur2007-marriage.pdf

(3) Viktor Vafeiadis. Concurrent separation logic and operational semantics. MFPS, 2011. Available from: http://www.mpi-sws.org/~viktor/cslsound/

(4) Xinyu Feng. Local Rely-Guarantee. POPL, 2009. Available from: http://staff.ustc.edu.cn/~xyfeng/research/publications/LRG.html

Clear and comprehensive coverage of RGSep.
NB: §3.2 and §3.3 are deprecated; read (3) instead.

Same as (1), but more dense

Simple soundness proof of CSL and RGSep

Summary

- RGSep as Concurrent Separation Logic + Rely-Guarantee
- Local state and shared state
- Boxed assertions, new definition of $*$
- Verification of a fine-grained concurrent datastructure
- Encoding CSL and RG
- Extensions and further work