

Idiom-Based Verification of Highly Concurrent Data Structures using Temporal Separation Logic

Alexey Gotsman

Noam Rinetzky

Hongseok Yang

IMDEA, Spain

Tel Aviv University, Israel

University of Oxford, UK

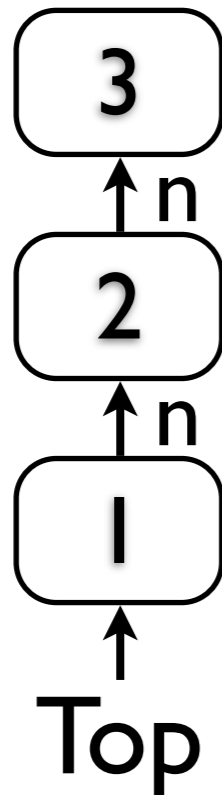
agenda

- verification problem: memory safety of highly concurrent data structures
- temporal idioms in verification problem
- temporal separation logic
 - Hoare Logic
 - Rely/Guarantee reasoning
 - Separation Logic
 - RGSep

Treiber's Stack (with GC)

```
Type Node {  
    int d  
    Node* n  
}
```

```
Node* Top
```



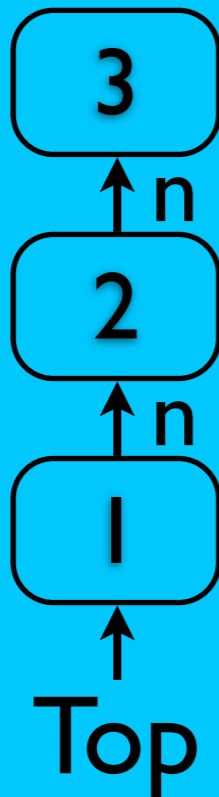
```
pop() {  
    L1: t = Top  
    s = t.n  
    if (!CAS(&Top, t, s))  
        goto L1  
}
```

```
push(k) {  
    x = new Node(k)  
    L2: t = Top  
    x.n = t  
    if (!CAS(&Top, t, x))  
        goto L2  
}
```

Treiber's Stack (with GC)

```
Type Node {  
    int d  
    Node* n  
}
```

Node* Top



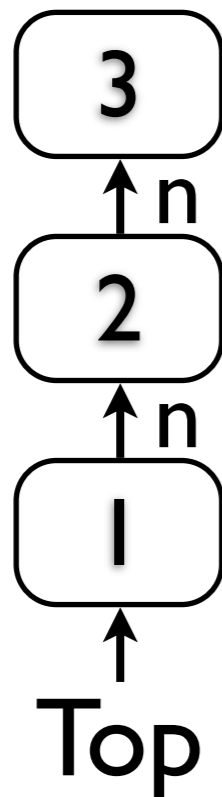
```
pop() {  
    L1: t = Top  
    s = t.n  
    if (!CAS(&Top, t, s))  
        goto L1  
}
```

```
push(k) {  
    x = new Node(k)  
    L2: t = Top  
    x.n = t  
    if (!CAS(&Top, t, x))  
        goto L2  
}
```

Treiber's Stack (with GC)

```
Type Node {  
    int d  
    Node* n  
}
```

```
Node* Top
```



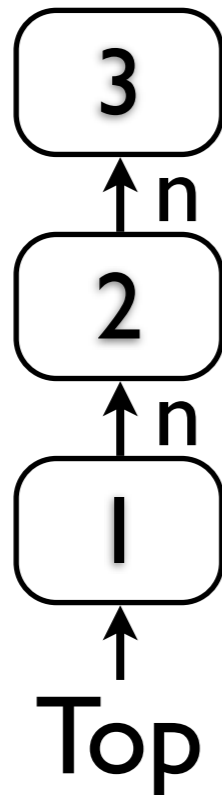
```
pop() {  
    L1: t = Top  
    s = t.n  
    if (!CAS(&Top, t, s))  
        goto L1  
}
```

```
push(k) {  
    x = new Node(k)  
    L2: t = Top  
    x.n = t  
    if (!CAS(&Top, t, x))  
        goto L2  
}
```

Treiber's Stack (with GC)

```
Type Node {  
    int d  
    Node* n  
}
```

Node* Top



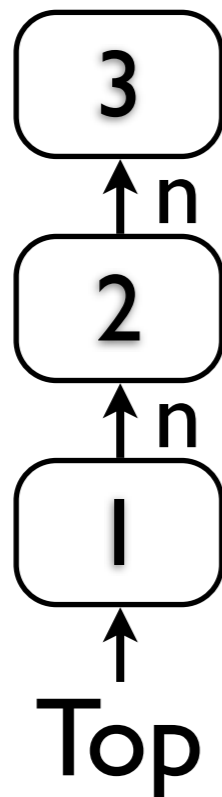
```
pop() {  
    L1: t = Top  
    s = t.n  
    if (!CAS(&Top, t, s))  
        goto L1  
}
```

```
push(k) {  
    x = new Node(k)  
    L2: t = Top  
    x.n = t  
    if (!CAS(&Top, t, x))  
        goto L2  
}
```

Treiber's Stack (with GC)

```
Type Node {  
    int d  
    Node* n  
}
```

Node* Top



```
pop() {  
    L1: t = Top  
    s = t.n  
    if (!CAS(&Top, t, s))  
        goto L1  
}
```

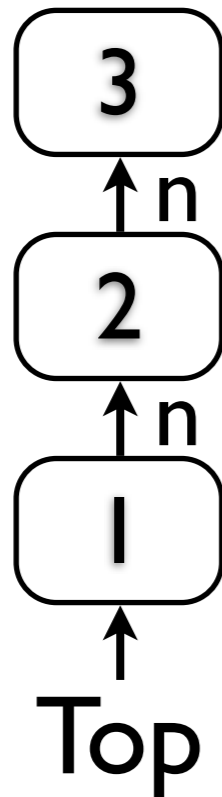
```
push(k) {  
    x = new Node(k)  
    L2: t = Top  
    x.n = t  
    if (!CAS(&Top, t, x))  
        goto L2  
}
```

```
CAS(&Top, t, s)  
if (Top == t) {  
    Top = s;  
    return true  
}  
else  
    return false;
```

Treiber's Stack (with GC)

```
Type Node {  
    int d  
    Node* n  
}
```

Node* Top



```
pop() {  
    L1: t = Top  
    s = t.n  
    if (!CAS(&Top, t, s))  
        goto L1  
}
```

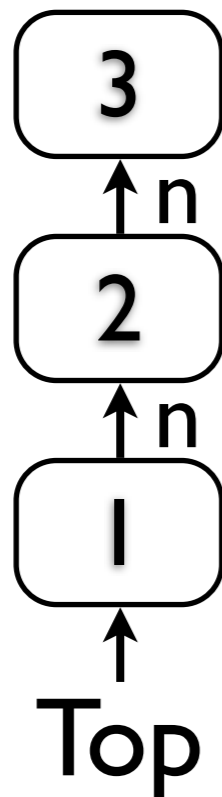
```
push(k) {  
    x = new Node(k)  
    L2: t = Top  
    x.n = t  
    if (!CAS(&Top, t, x))  
        goto L2  
}
```

Treiber's Stack (with GC)

```
Type Node {  
    int d  
    Node* n  
}
```

```
Node* Top
```

```
pop() {  
    L1: t = Top  
    s = t.n  
    if (!CAS(&Top, t, s))  
        goto L1  
}
```



```
push(k) {  
    x = new Node(k)  
    L2: t = Top  
    x.n = t  
    if (!CAS(&Top, t, x))  
        goto L2  
}
```

races by design

t1

L1: t=Top

s=t.n

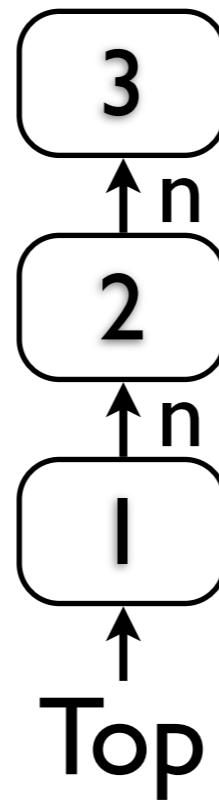
if !CAS(&Top, t, s)
goto 1:

t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1



races by design

t1



L1: t=Top

s=t.n

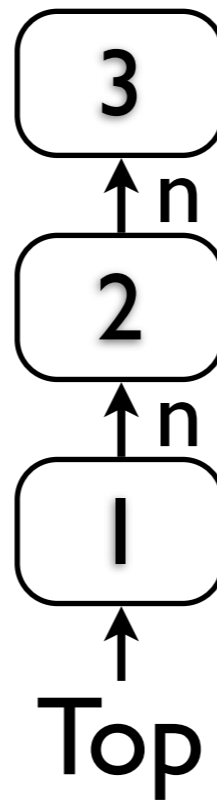
if !CAS(&Top, t, s)
goto 1:

t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1



races by design

t1

L1: t=Top



s=t.n

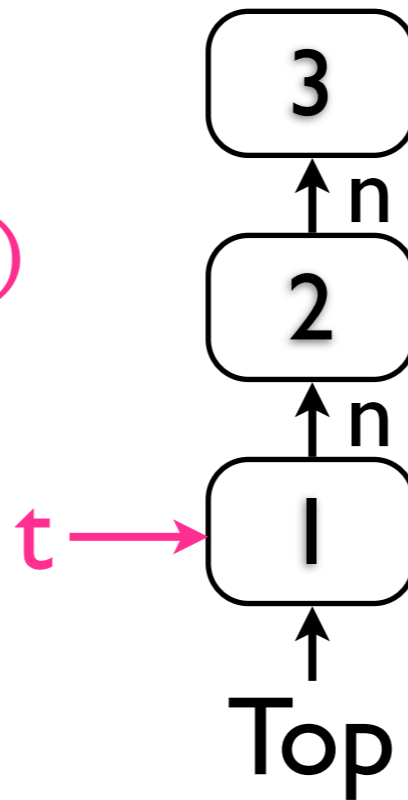
if !CAS(&Top, t, s)
goto 1:

t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1



races by design

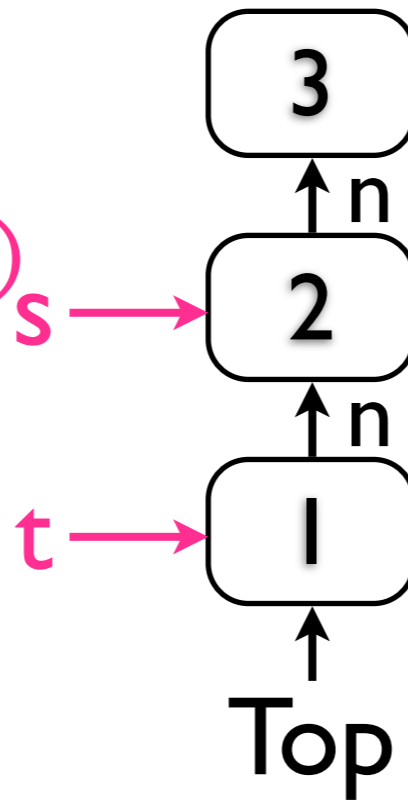
t1

L1: t=Top

s=t.n



if !CAS(&Top, t, s)
goto 1:



t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1

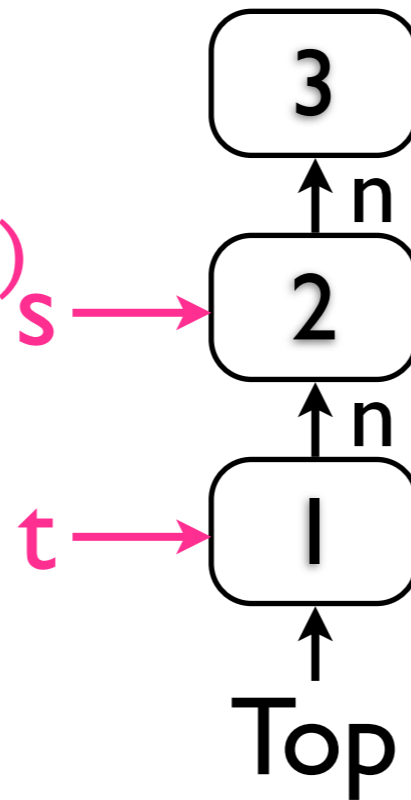
races by design

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:



t2



L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1

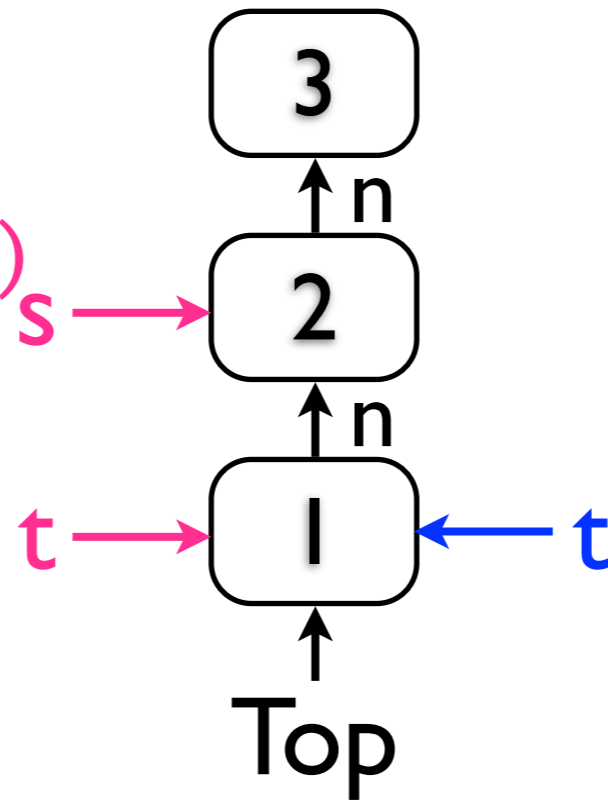
races by design

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:



t2

L1: t=Top



s=t.n

if !CAS(&Top, t, s)
goto L1

races by design

t1

L1: t=

s=t.n

if !CAS(&Top, t, s)
goto 1:

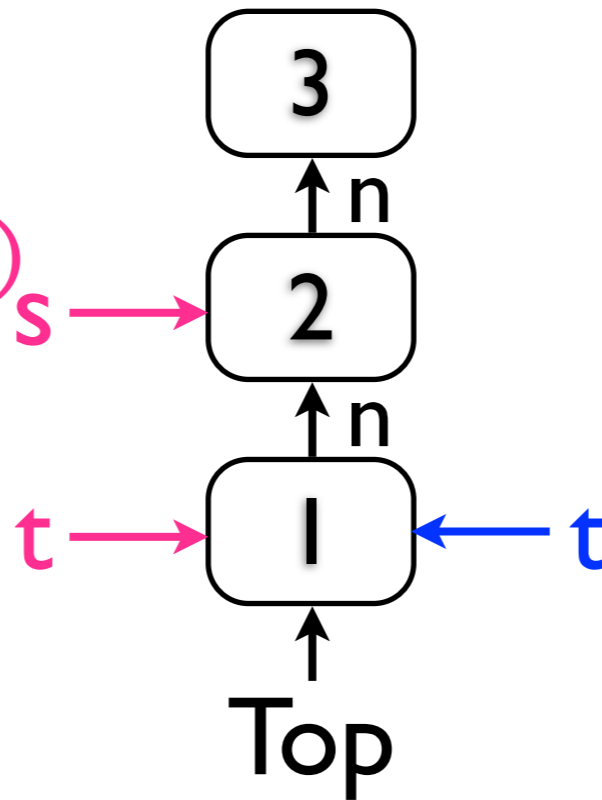
```
CAS(&Top, t, s)
if (Top==t) {
  Top=s;
  return true
}
else
  return false;
```

t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1



races by design

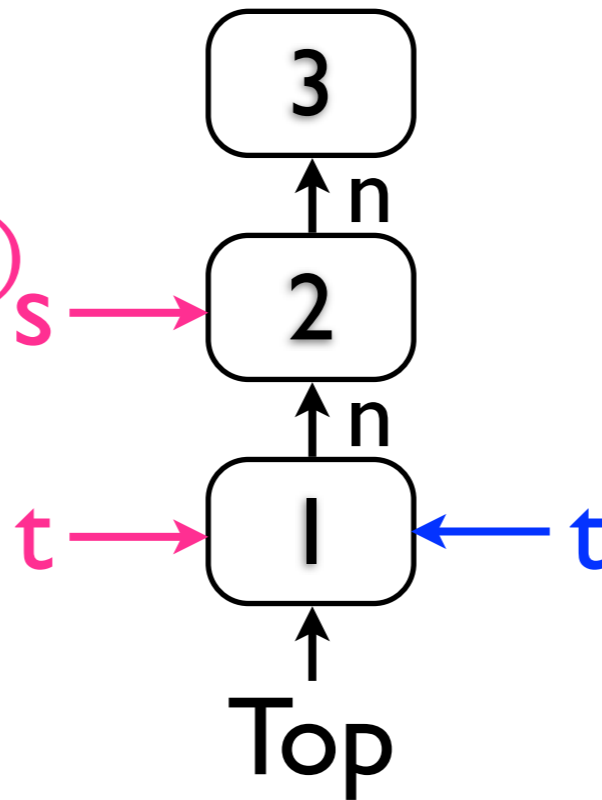
t1

L1: t=Top

s=t.n



if !CAS(&Top, t, s)
goto 1:



t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1

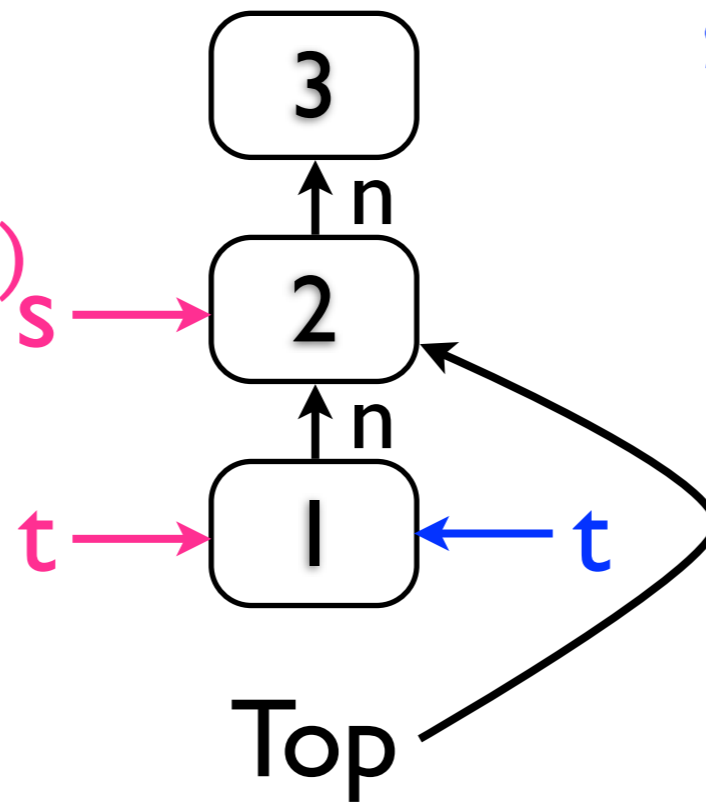
races by design

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:



t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1

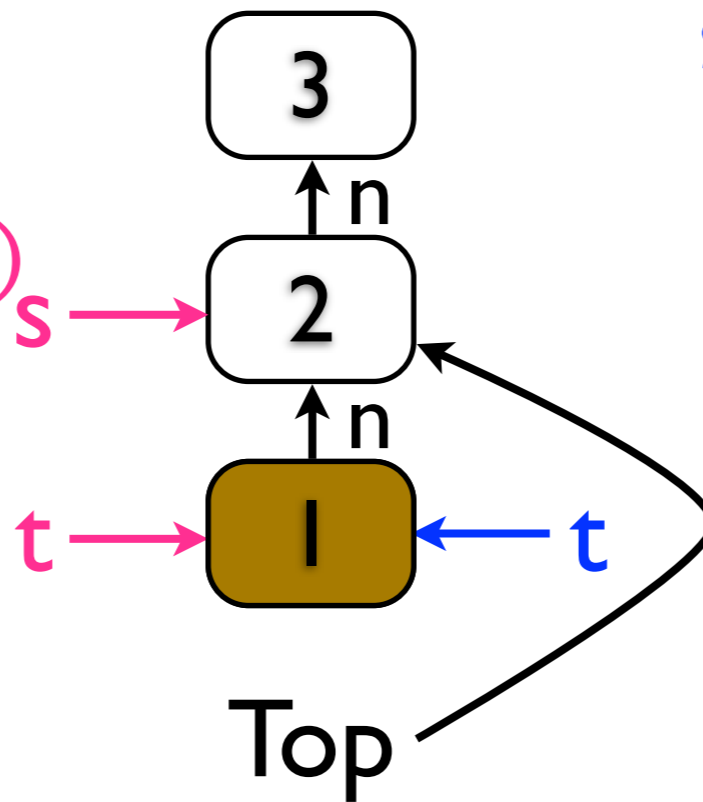
races by design

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:



t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1

races by design

t1

L1: t=Top

s=t.n

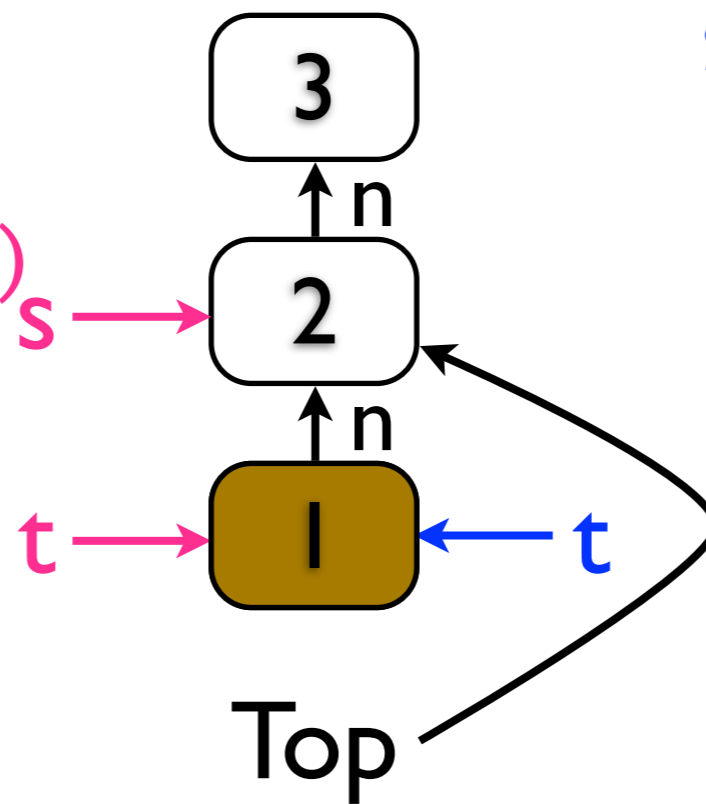
if !CAS(&Top, t, s)
goto 1:

t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1



races by design

t1

L1: t=Top

s=t.n

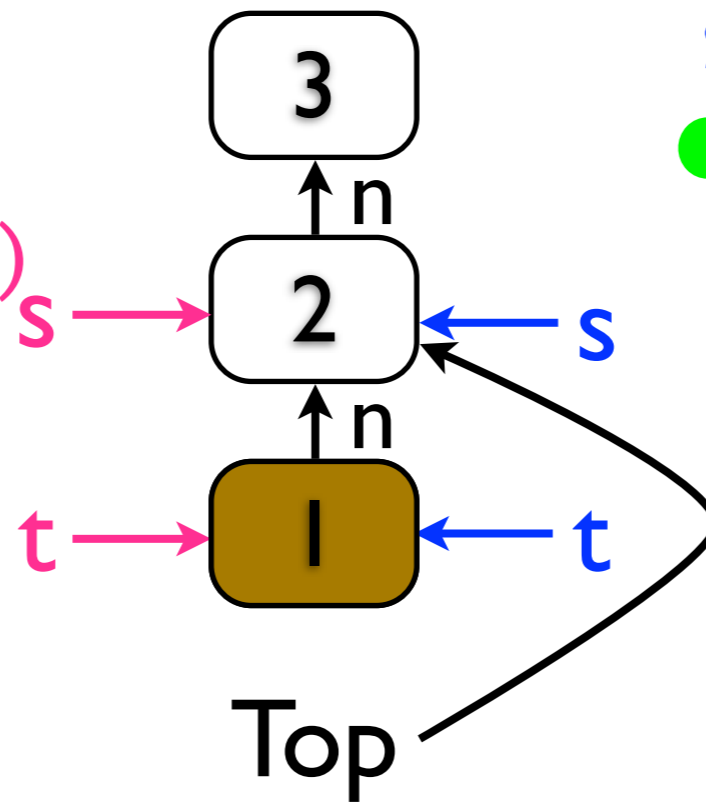
if !CAS(&Top, t, s)
goto 1:

t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1



races by design

t1

L1: t=Top

s=t.n

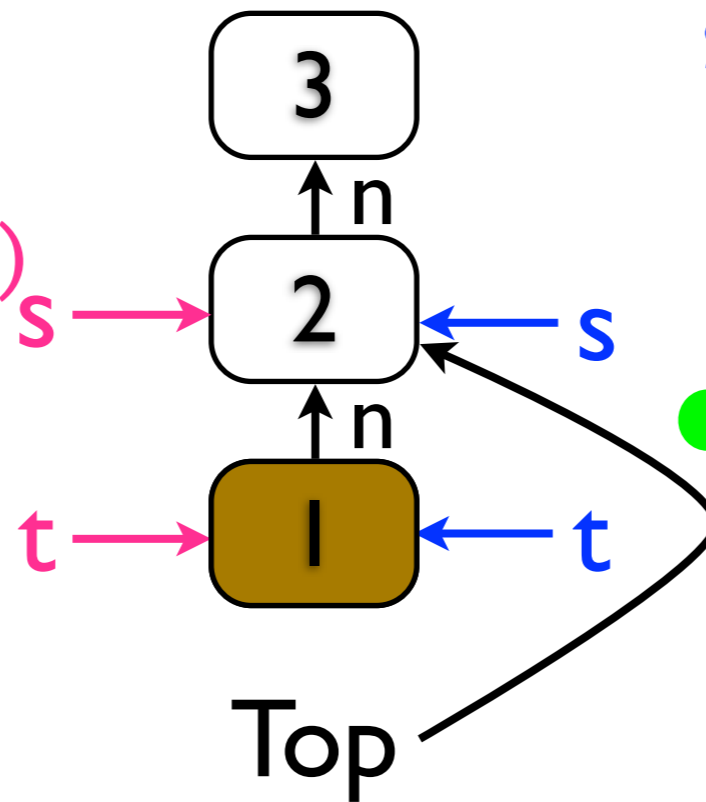
if !CAS(&Top, t, s)
goto 1:

t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1



races by design

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:

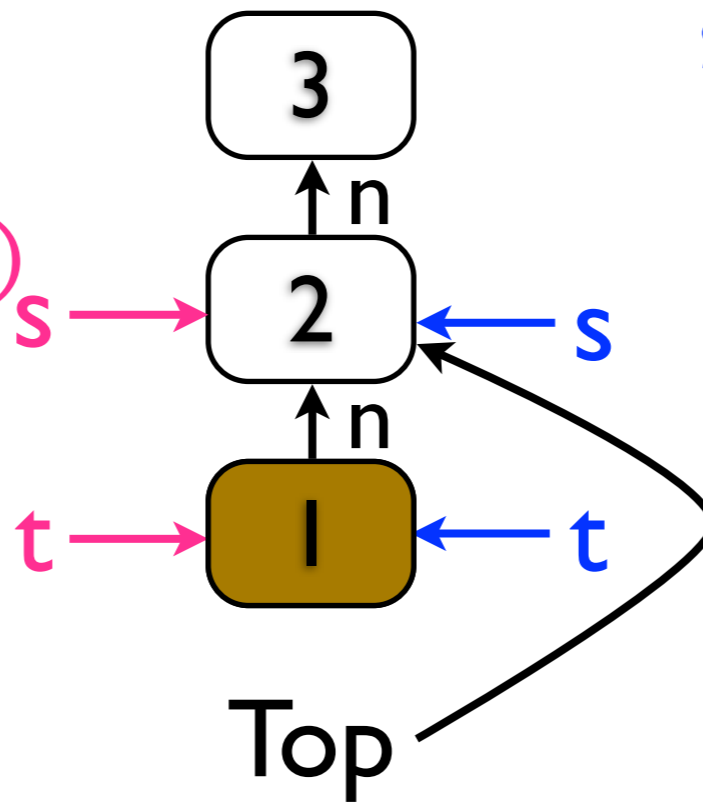
t2



L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1



races by design

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:

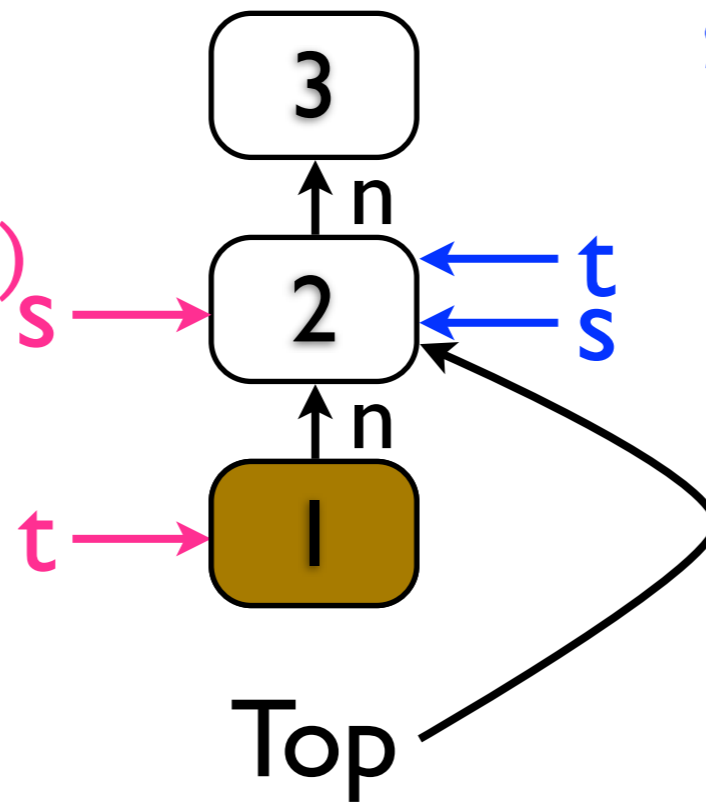
t2

L1: t=Top



s=t.n

if !CAS(&Top, t, s)
goto L1



races by design

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:

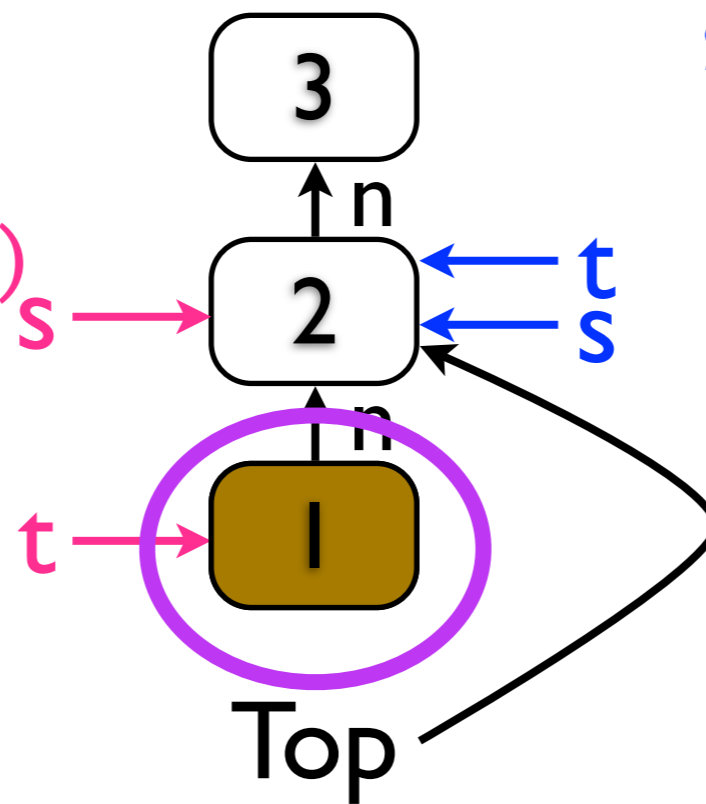
t2

L1: t=Top



s=t.n

if !CAS(&Top, t, s)
goto L1



races by design

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:

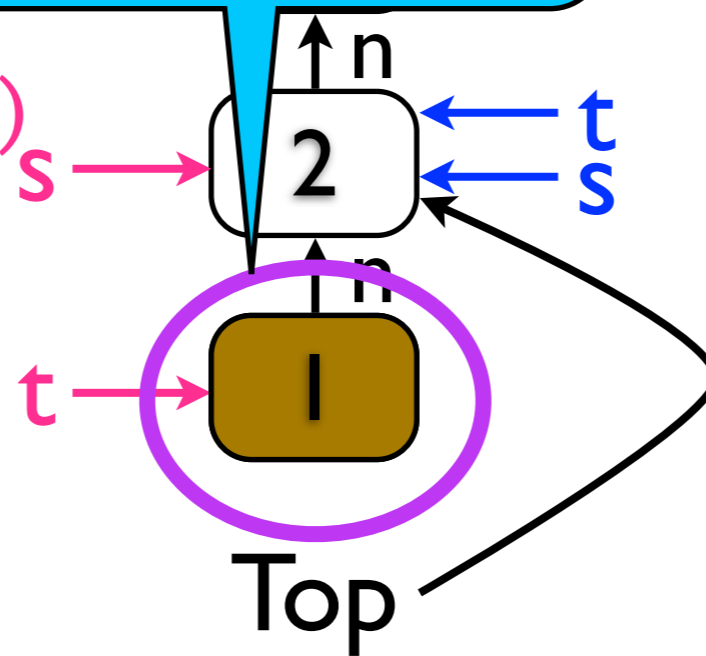
t2

L1: t=Top

●
s=t.n

if !CAS(&Top, t, s)
goto L1

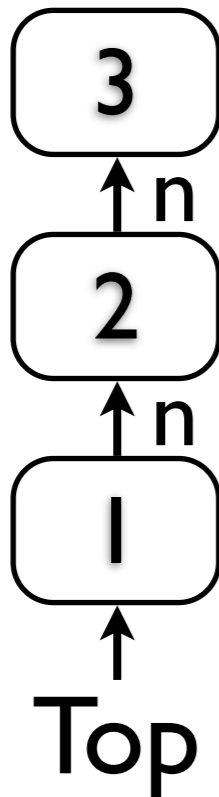
no problem:
access to removed
objects



Treiber's Stack (no GC)

```
Type Node {  
    int d  
    Node* n  
}
```

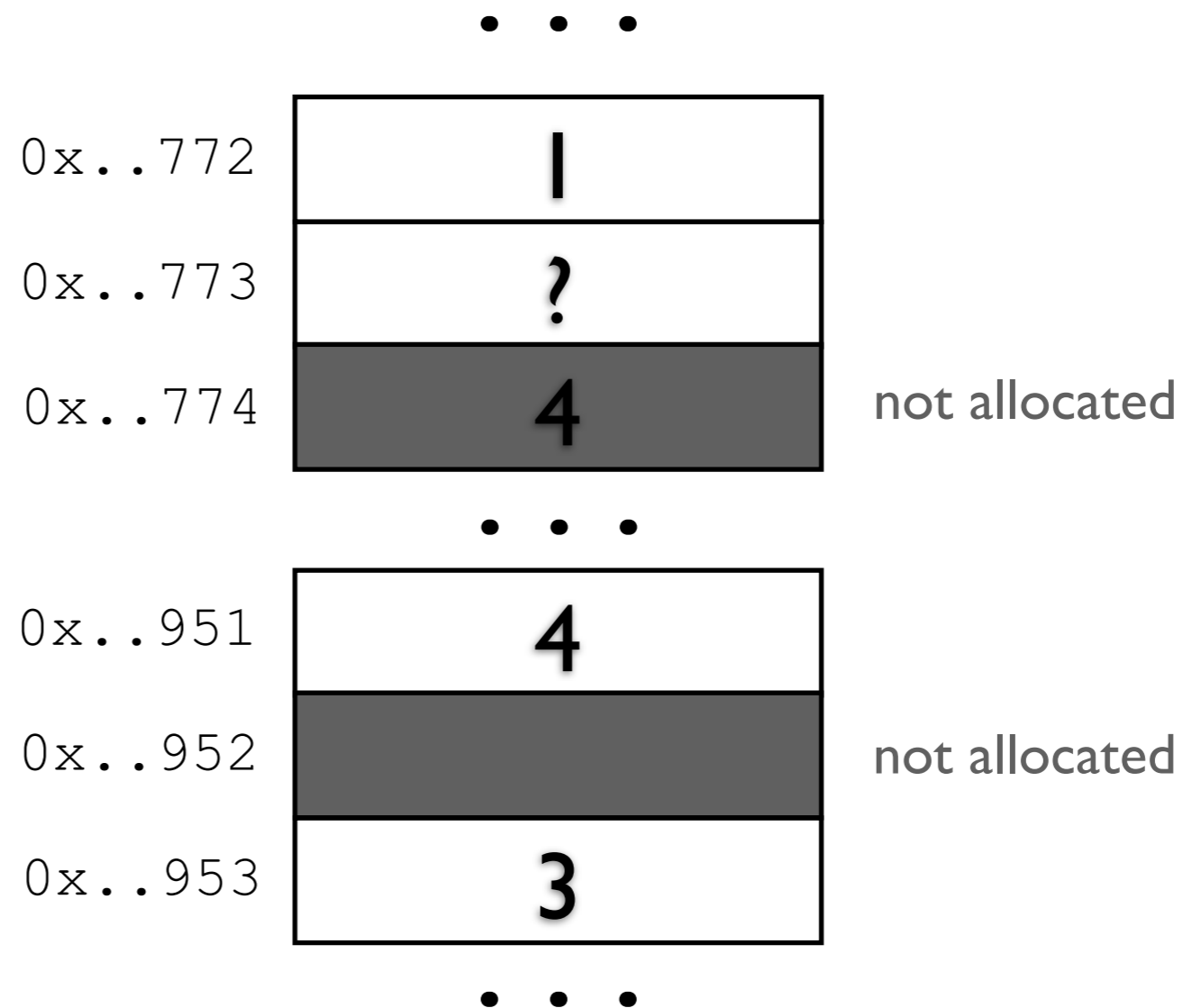
```
Node* Top
```



```
pop() {  
    L1: t = Top  
    s = t.n  
    if (!CAS(&Top, t, s))  
        goto L1  
    free(t)  
}  
  
push(k) {  
    x = new Node(k)  
    L2: t = Top  
    x.n = t  
    if (!CAS(&Top, t, x))  
        goto L2  
}
```

memory safety

- only allocated memory locations are accessed



racing for free is hazardous

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:

free(t)

t2

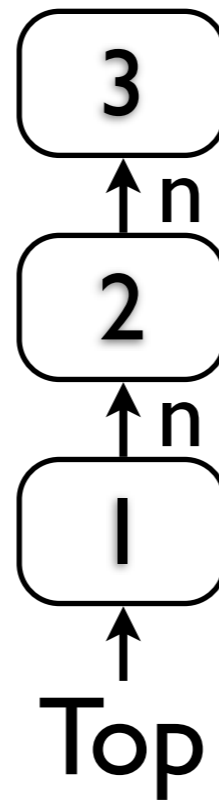
L1: t=Top



s=t.n

if !CAS(&Top, t, s)
goto L1

free(t)



racing for free is hazardous

t1

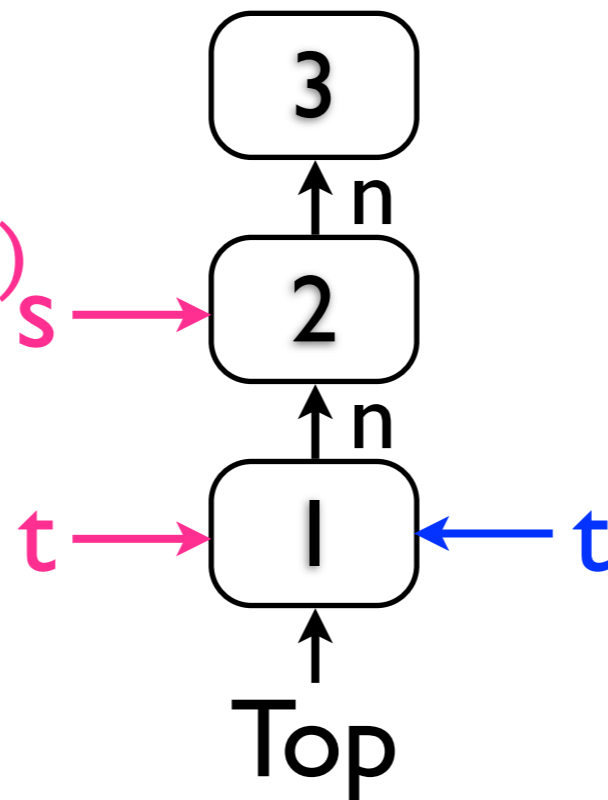
L1: t=Top

s=t.n



if !CAS(&Top, t, s)
goto 1:

free(t)



t2

L1: t=Top



s=t.n

if !CAS(&Top, t, s)
goto L1

free(t)

racing for free is hazardous

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:

free(t)

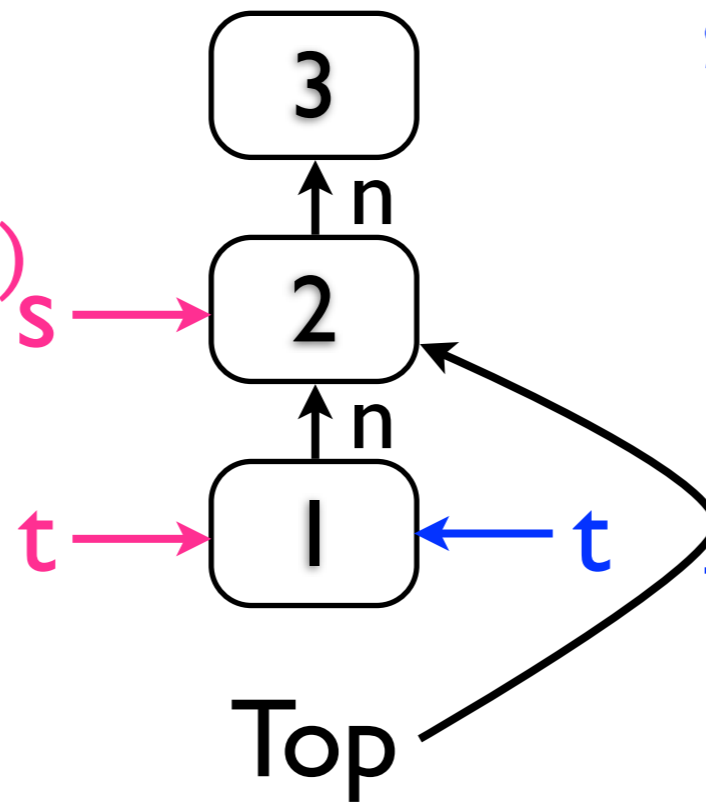
t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1

free(t)



racing for free is hazardous

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:

free(t)

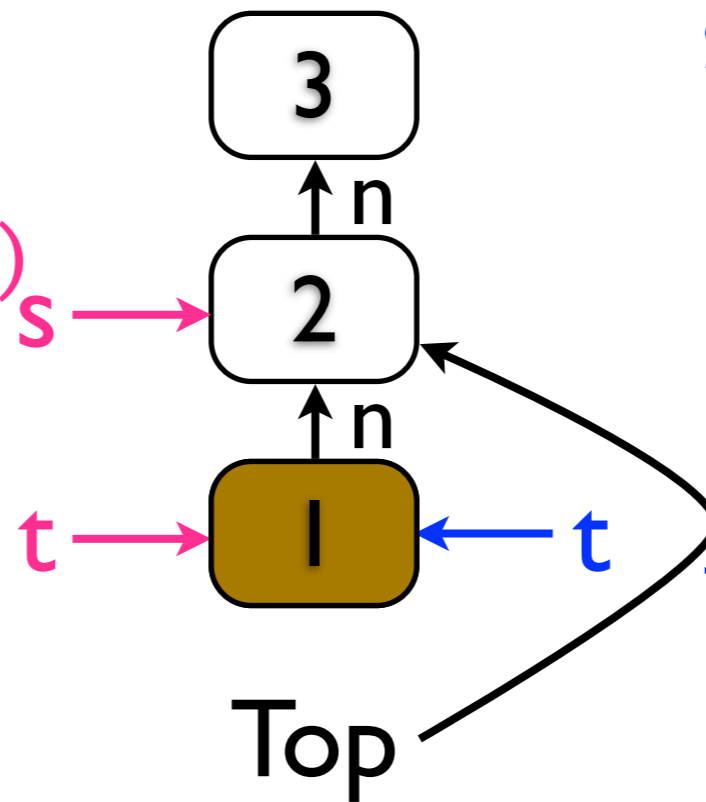
t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1

free(t)



racing for free is hazardous

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:

free(t)

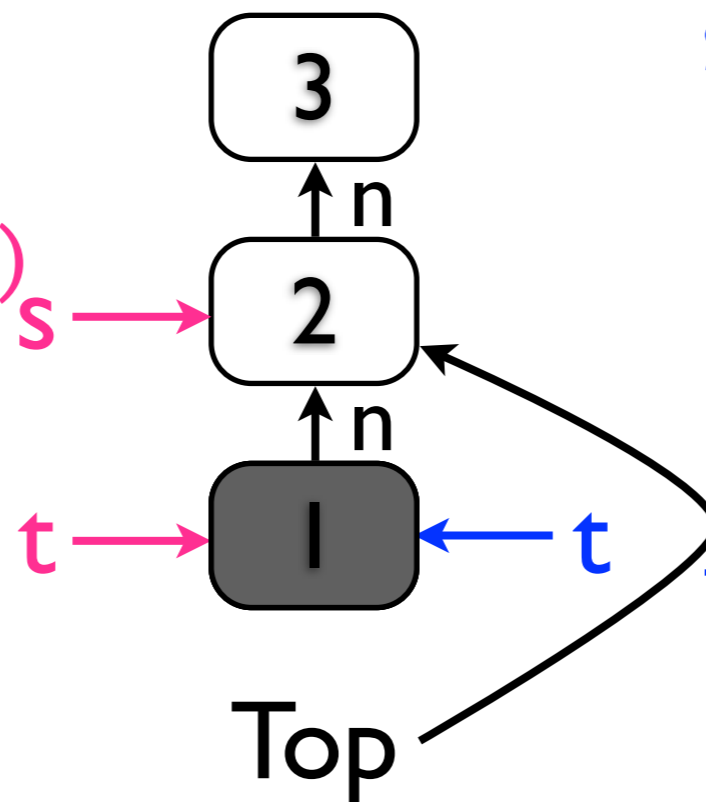
t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1

free(t)



racing for free is hazardous

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:

free(t)

t2

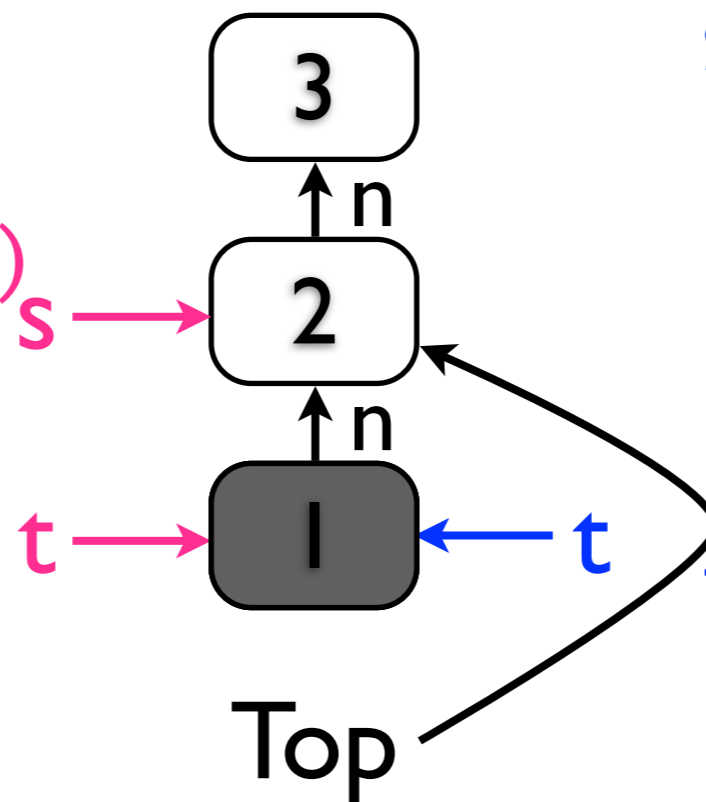
L1: t=Top



s=t.n

if !CAS(&Top, t, s)
goto L1

free(t)



racing for free is hazardous

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:

free(t)

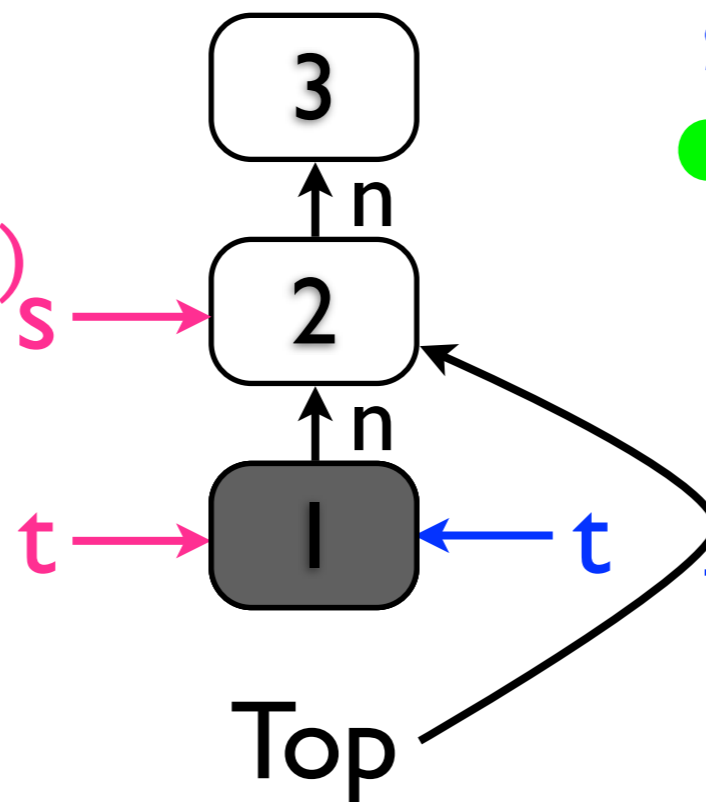
t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1

free(t)



racing for free is hazardous

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:

free(t)

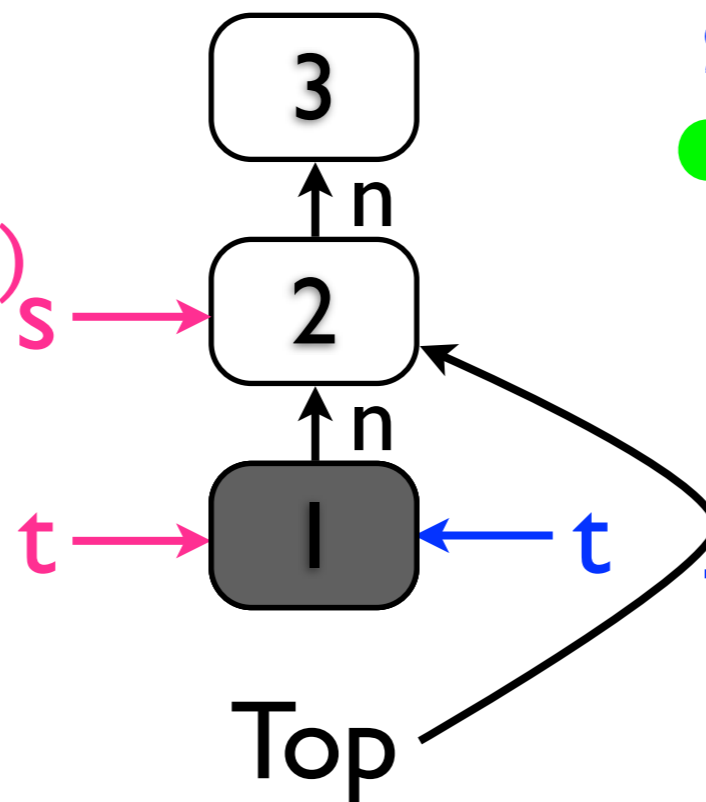
t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1

free(t)



racing for free is hazardous

t1

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto 1:

free(t)

problem:
access to removed
objects

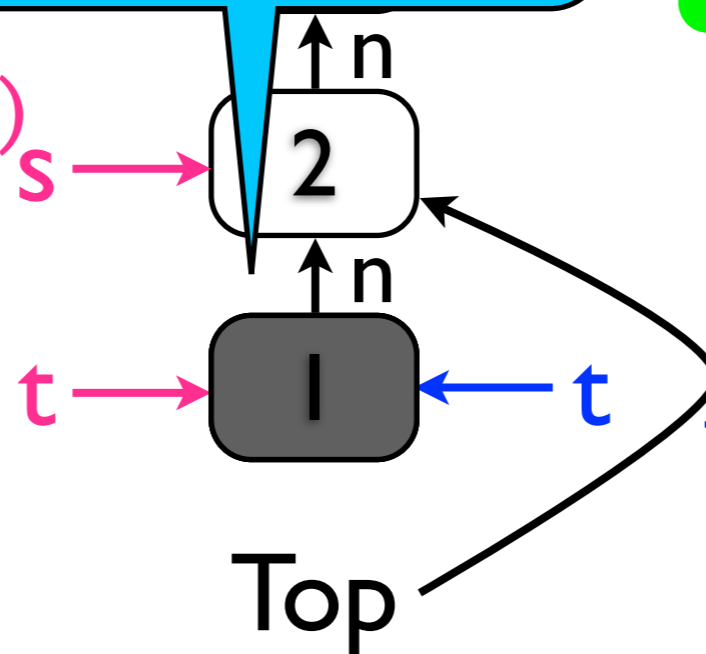
t2

L1: t=Top

s=t.n

if !CAS(&Top, t, s)
goto L1

free(t)



programming challenge

- ensuring memory safety
 - shared memory concurrency
 - lock free data structures
 - no GC

programming solutions

- **RCU** [McKenney Slingwine'98]
- **Hazard Pointers** [Michael'02]
- **Pass the Buck** [Herlihy Luchangco Moir'02]
- **Epoch** [Fraser Haris'03]

programming solutions

- **RCU** [McKenney Slingwine'98]
- **Hazard Pointers** [Michael'02]
- **Pass the Buck** [Herlihy Luchangco Moir'02]
- **Epoch** [Fraser Haris'03]
- programming methodologies
- per data structure specialization

hazard methodology

- add global “hazard” *pointers*
- “protect” objects using hazard pointers
- check hazard pointers before calling `free`

if a hazard pointer points to an object continuously since the object is “safe” (in the data structure) then the object is not reclaimed

Treiber's stack + H.P.

```
Type Node {  
    int d  
    Node* n  
}
```

```
Node* Top
```

```
Node* HP[NTID]
```

```
pop() {  
    L1: t = Top
```

```
    HP[tid] = t
```

```
    if (t != Top) goto L1
```

```
    s = t.n
```

```
    if (!CAS(&Top, t, s))  
        goto L1
```

```
    retire(t)
```

```
}
```

Treiber's stack + H.P.

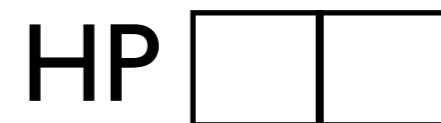
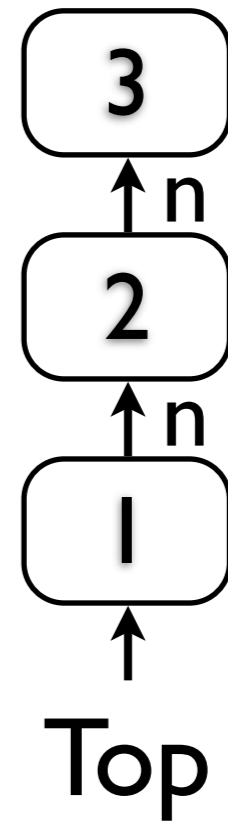
```
retire(r) {  
    i = 0;  
    HP[tid] = null  
    while (i < NTID) {  
        while (HP[i] == r)  
            skip  
        i++  
    }  
    free (r)  
}
```

```
pop() {  
    L1: t = Top  
    HP[tid] = t  
    if (t != Top) goto L1  
    s = t.n  
    if (!CAS(&Top, t, s))  
        goto L1  
    retire(t)  
}
```

“weird” races

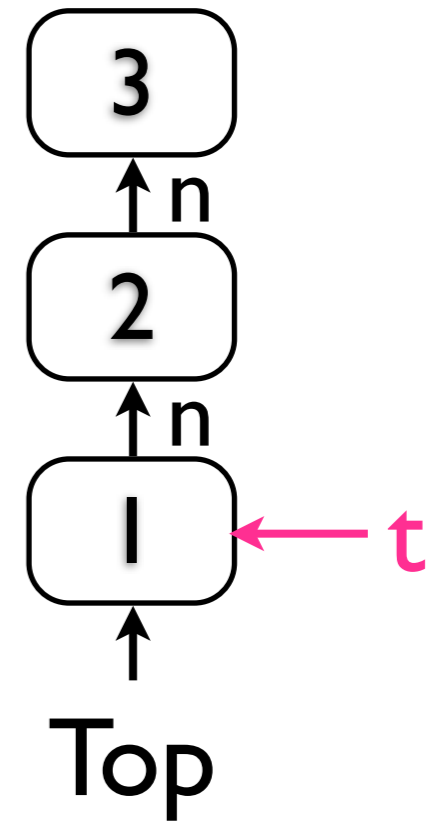
● ●

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



“weird” races

pop() {
 L1: t=Top

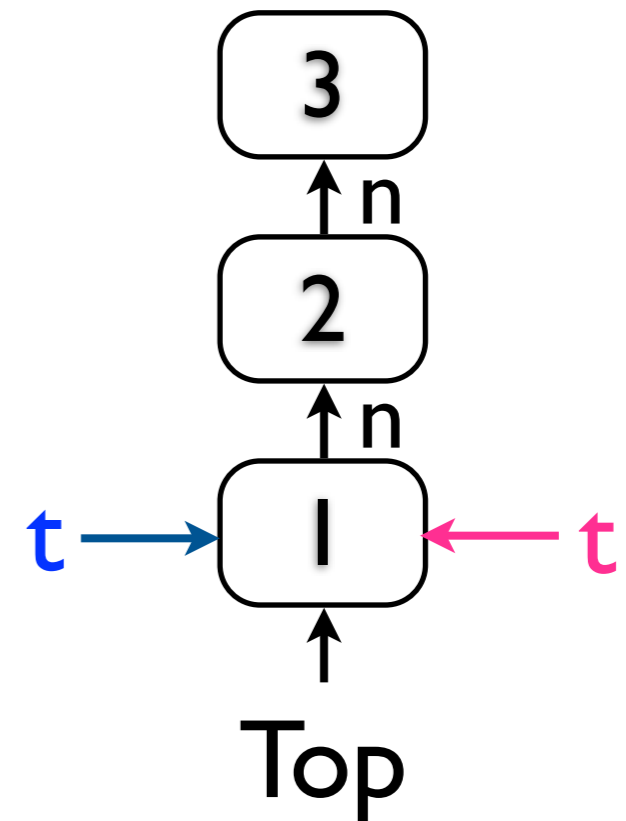
 HP[tid] = t

 if (t != Top) goto L1

 s=t.n

 if (!CAS(&Top, t, s))
 goto L1

 retire(t)
}

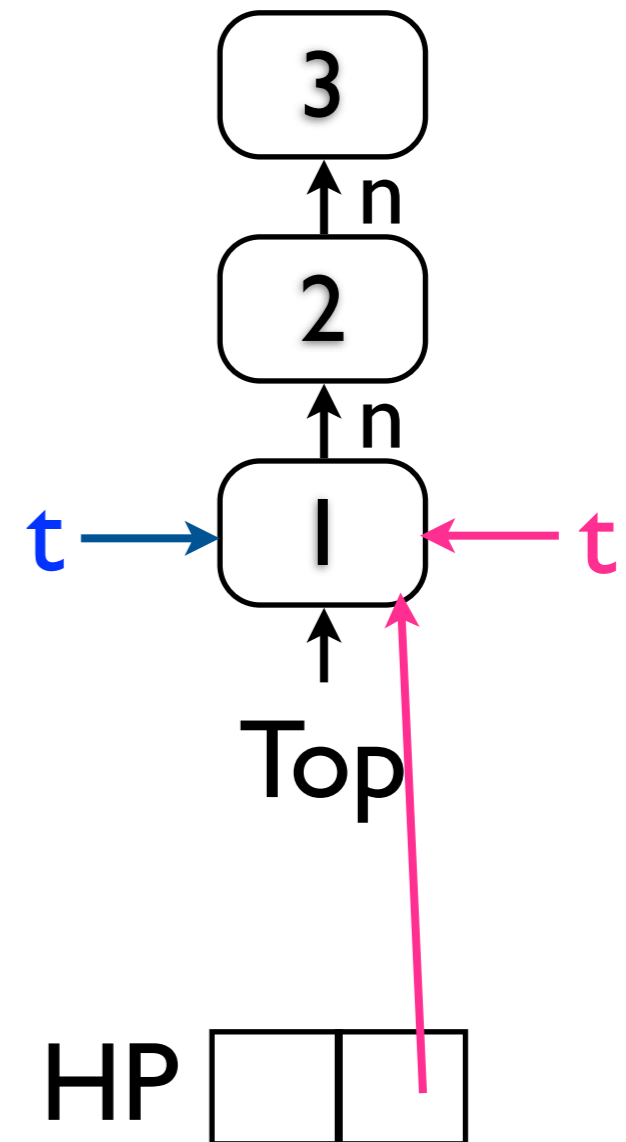


HP

--	--

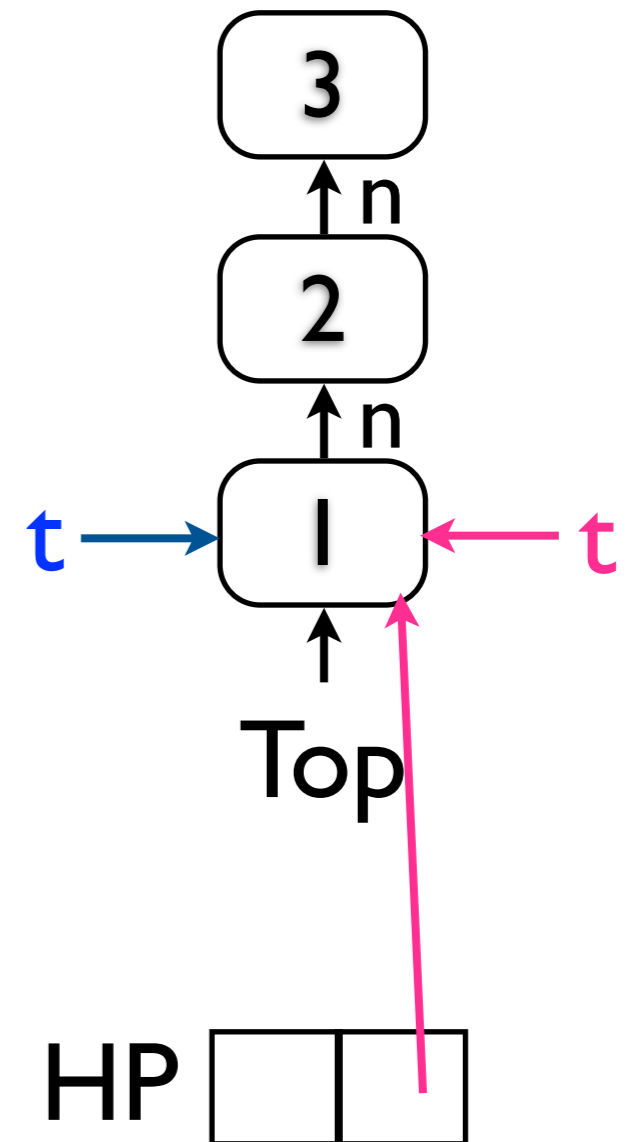
“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



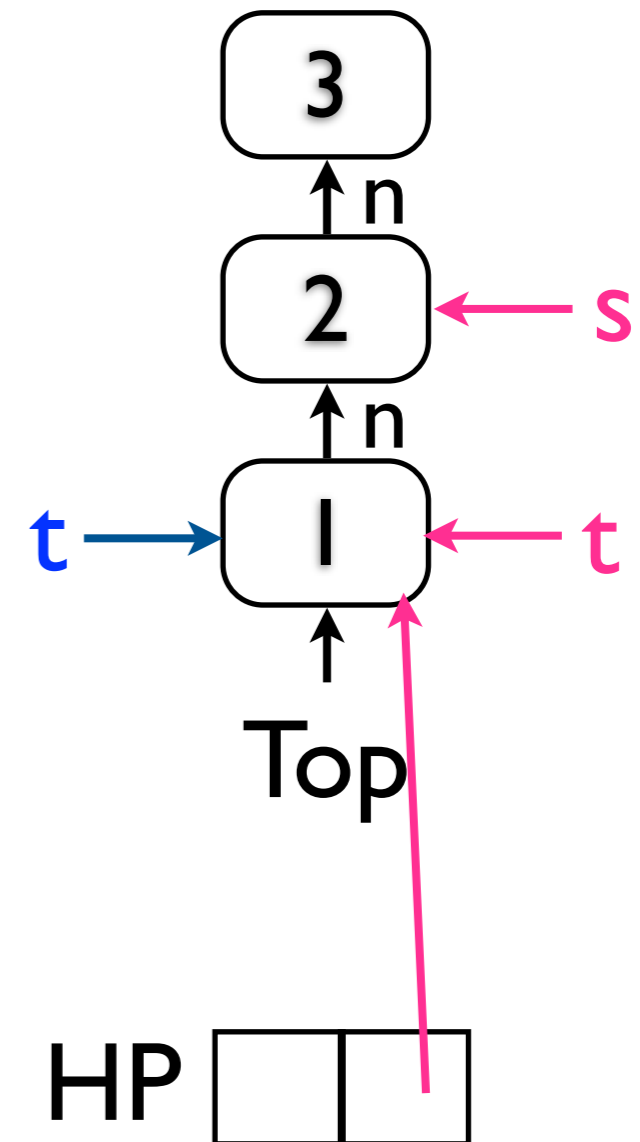
“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



“weird” races

```
pop() {
    L1: t=Top

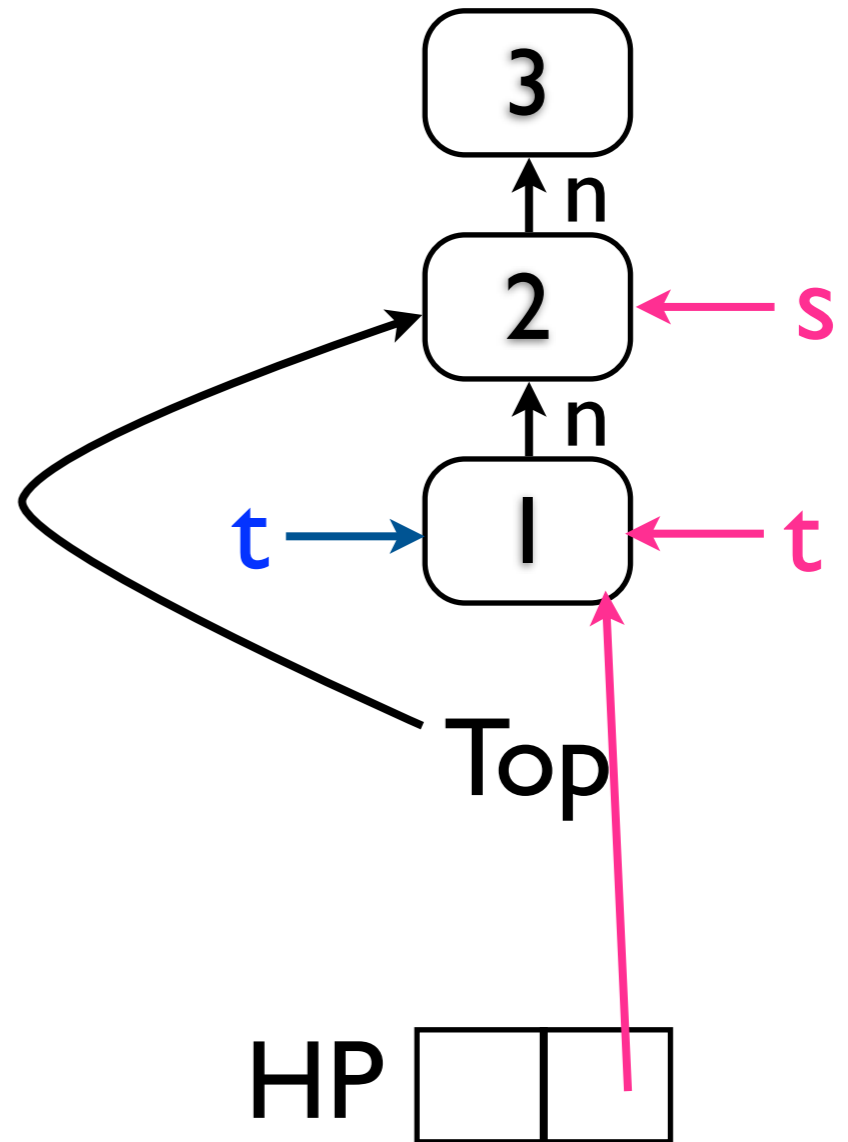
    HP[tid] = t

    if (t != Top) goto L1

    s=t.n

    if (!CAS(&Top,t,s))
        goto L1

    retire(t)
}
```



“weird” races

```
pop() {
    L1: t=Top

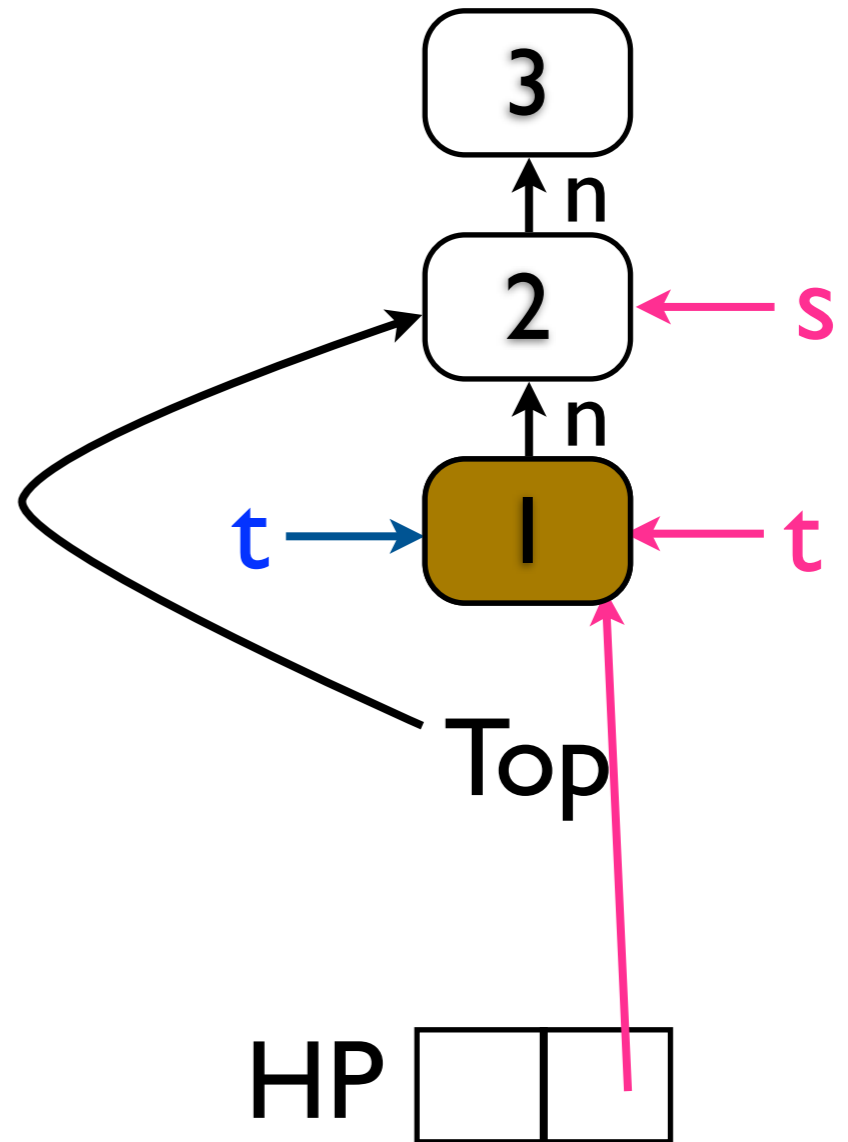
    HP[tid] = t

    if (t != Top) goto L1

    s=t.n

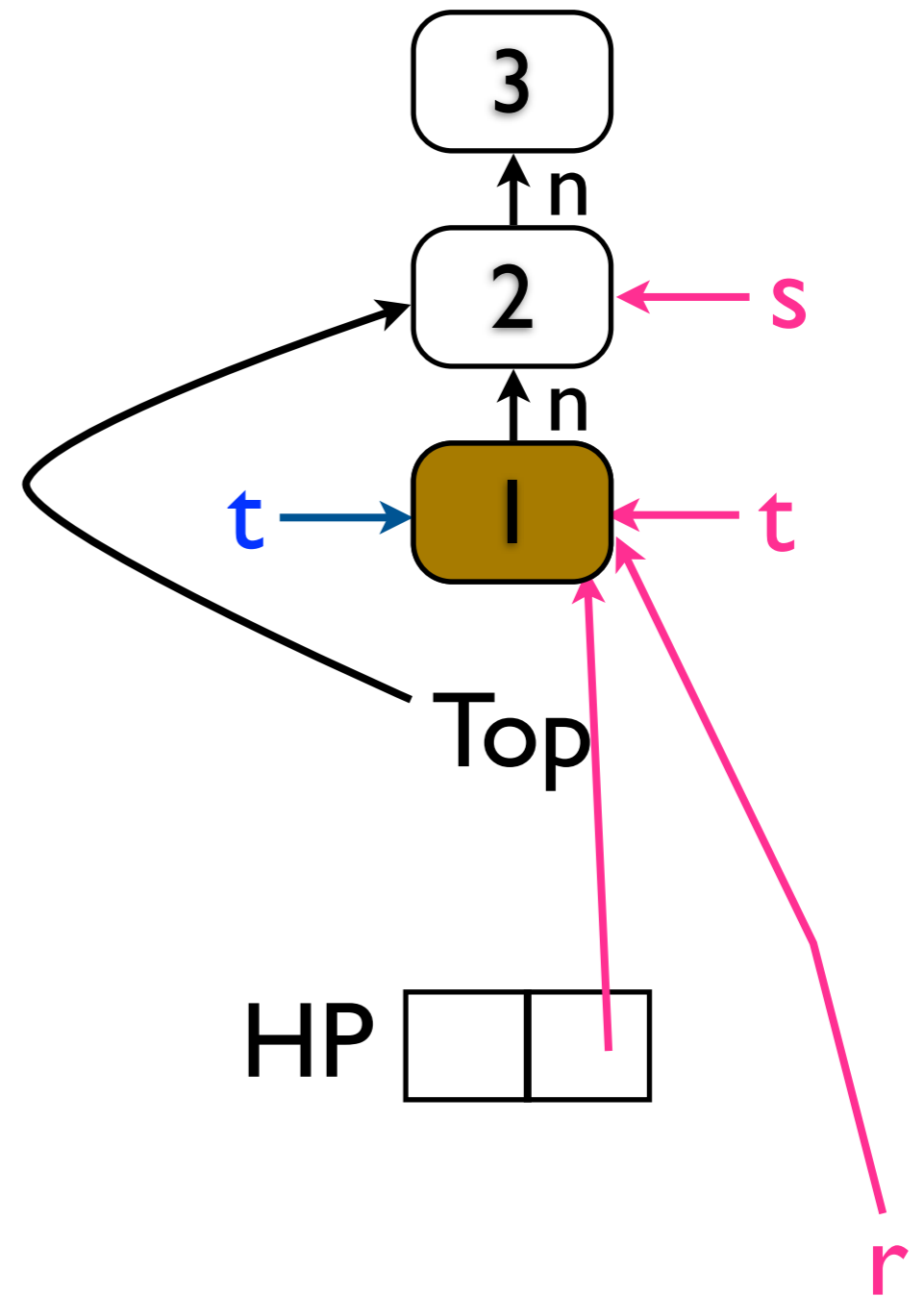
    if (!CAS(&Top,t,s))
        goto L1

    retire(t)
}
```



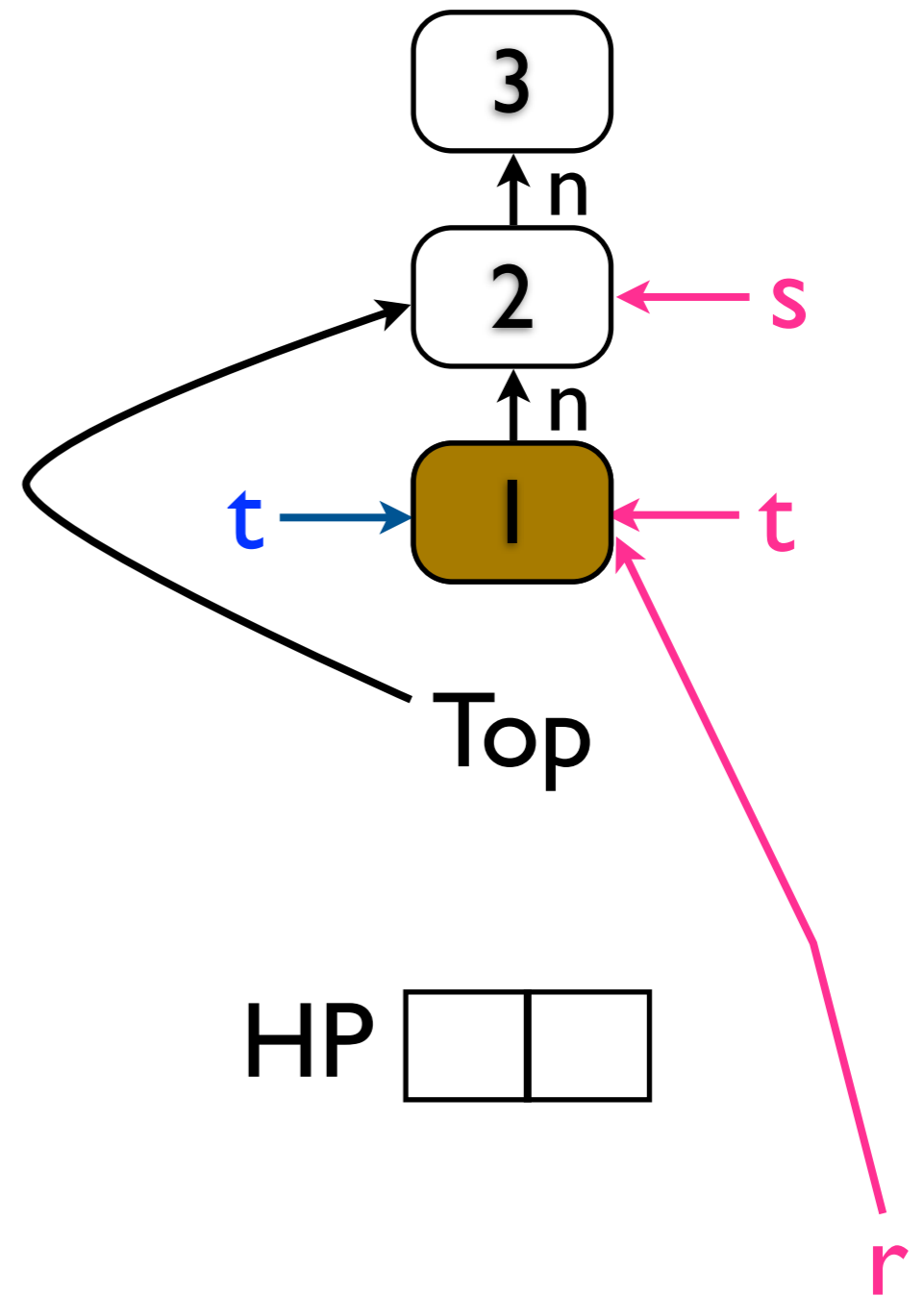
“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



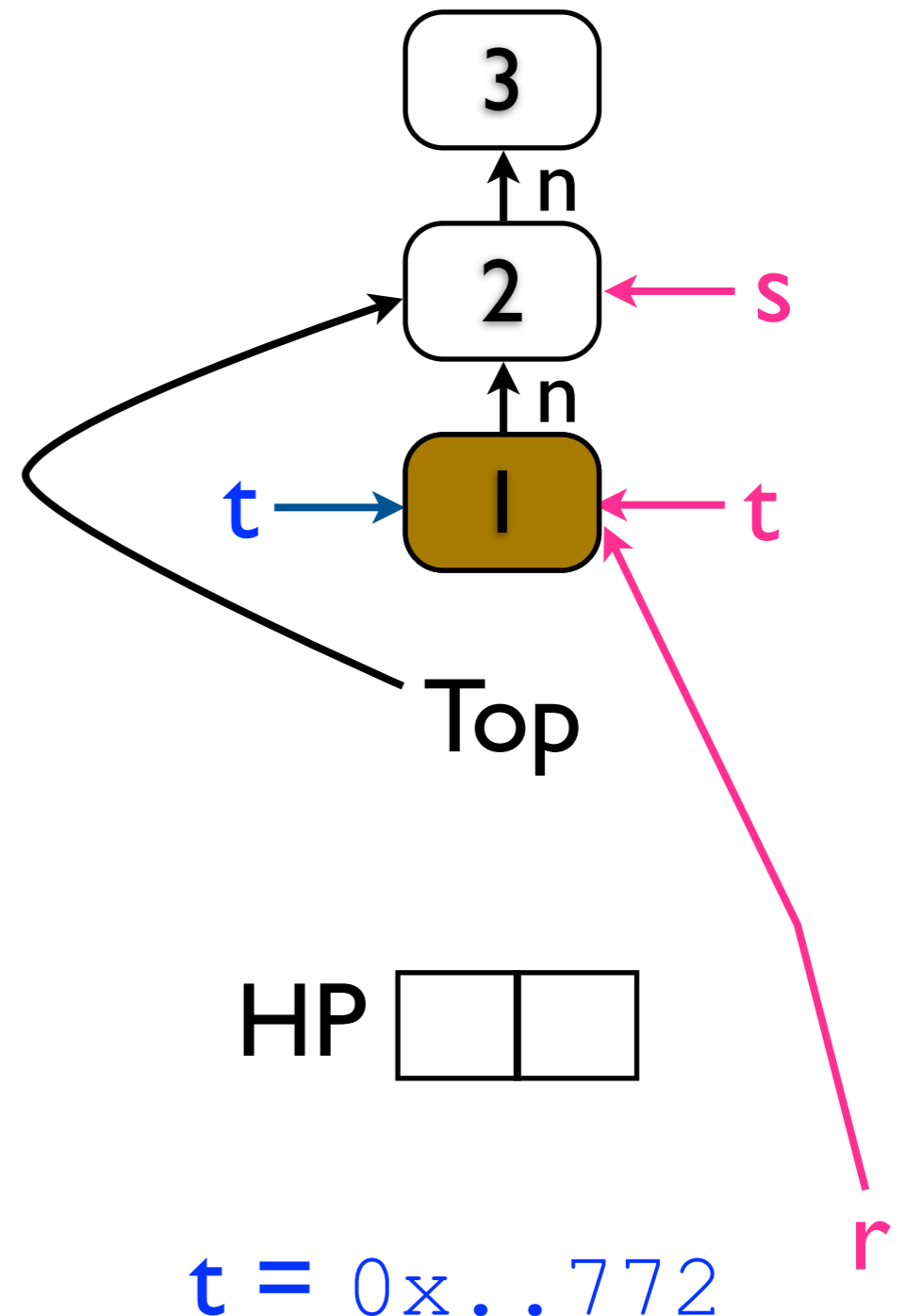
“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



“weird” races

```
pop() {
    L1: t=Top

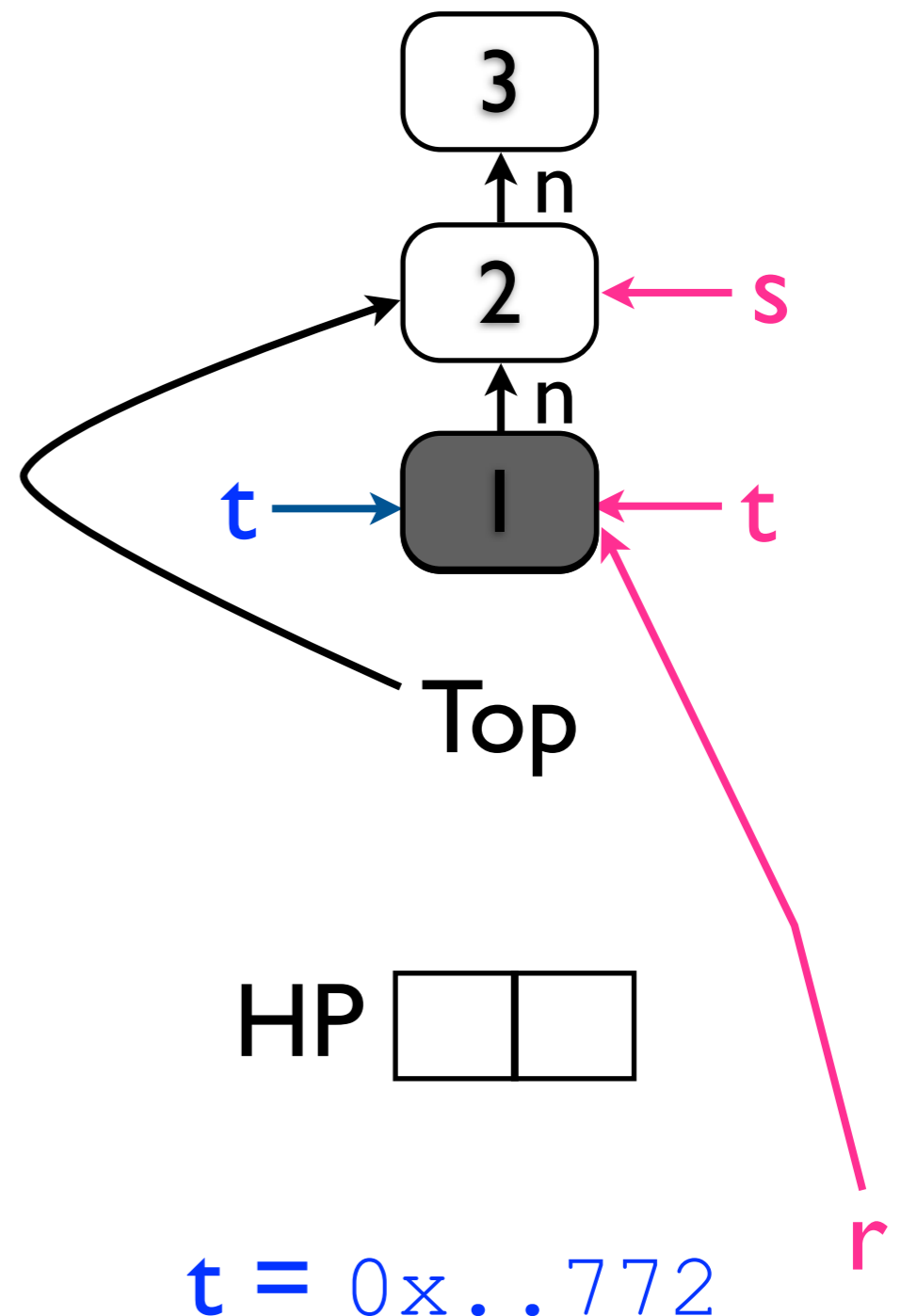
    HP[tid] = t

    if (t != Top) goto L1

    s=t.n

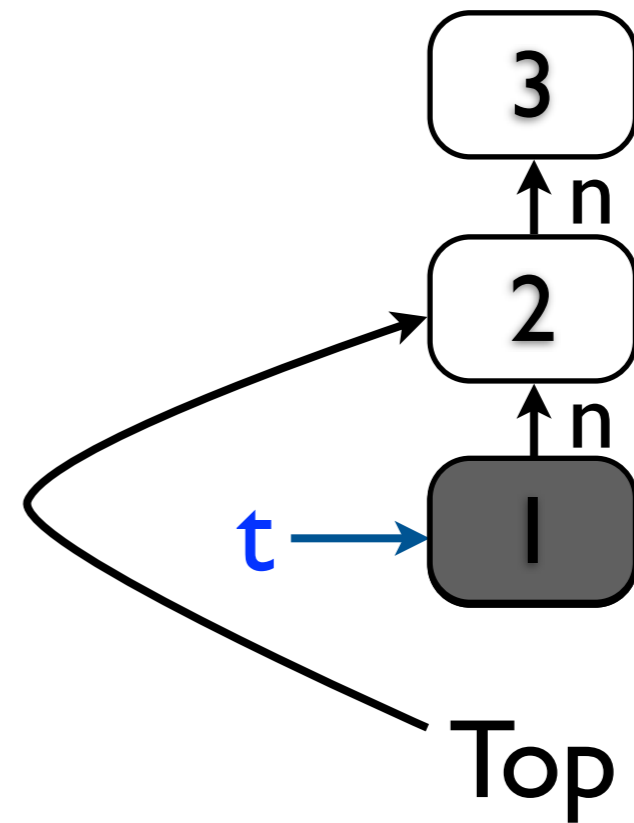
    if (!CAS(&Top,t,s))
        goto L1

    retire(t)
}
```



“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top,t,s))  
    goto L1  
  
  retire(t)  
}
```



HP

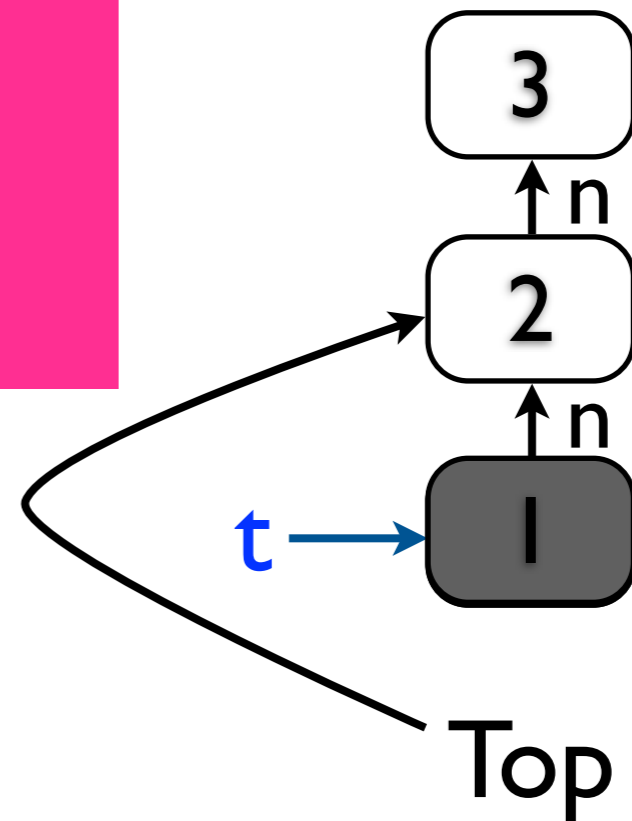
--	--

$t = 0x...772$

“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```

freeing a
reachable
object



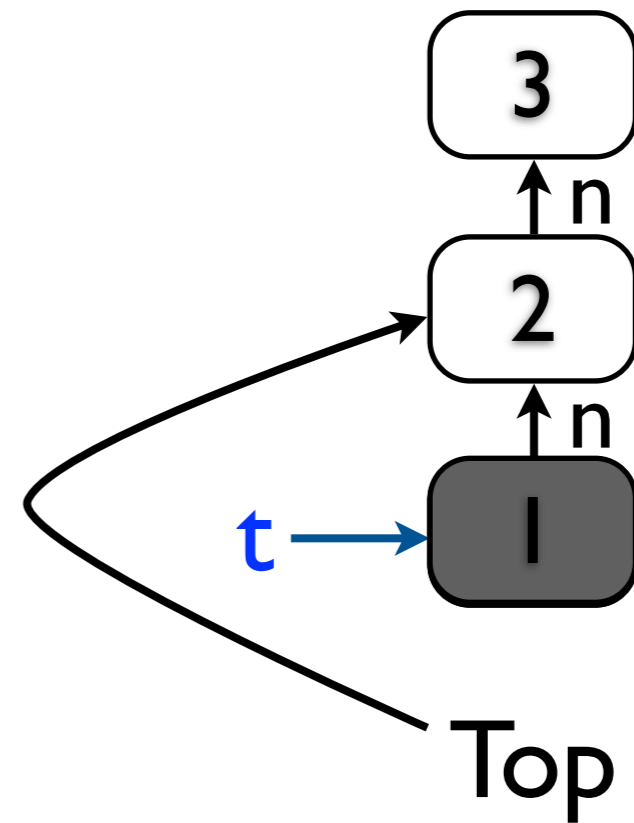
HP

--	--

$t = 0x..772$

“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top,t,s))  
    goto L1  
  
  retire(t)  
}
```



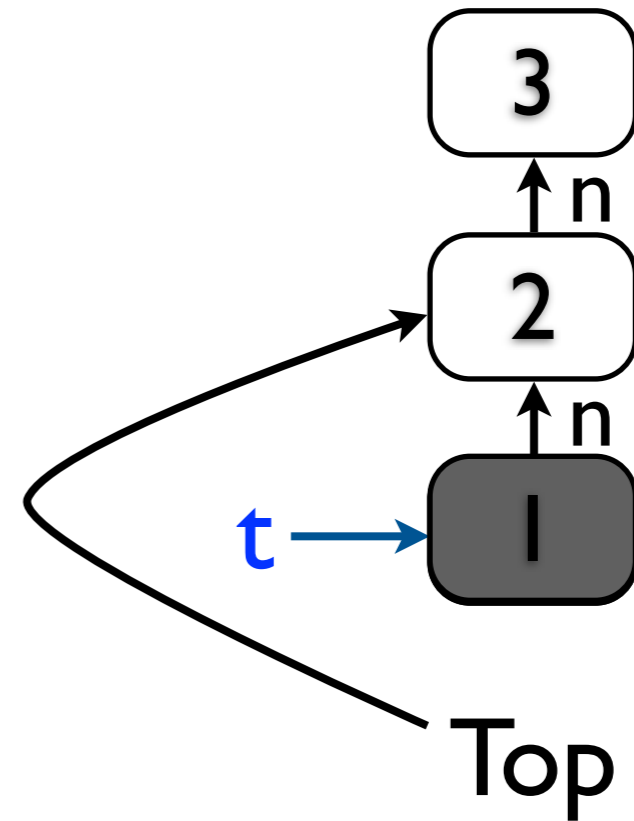
HP

--	--

$t = 0x...772$

“weird” races

```
pop() {  
  L1: t=Top  
push(k) {  
  x = new Node(k)  
  L2: t=Top  
  x.n = t  
  if (!CAS(&Top, t, x))  
    goto L2  
}  
  goto L1  
retire(t)  
}
```



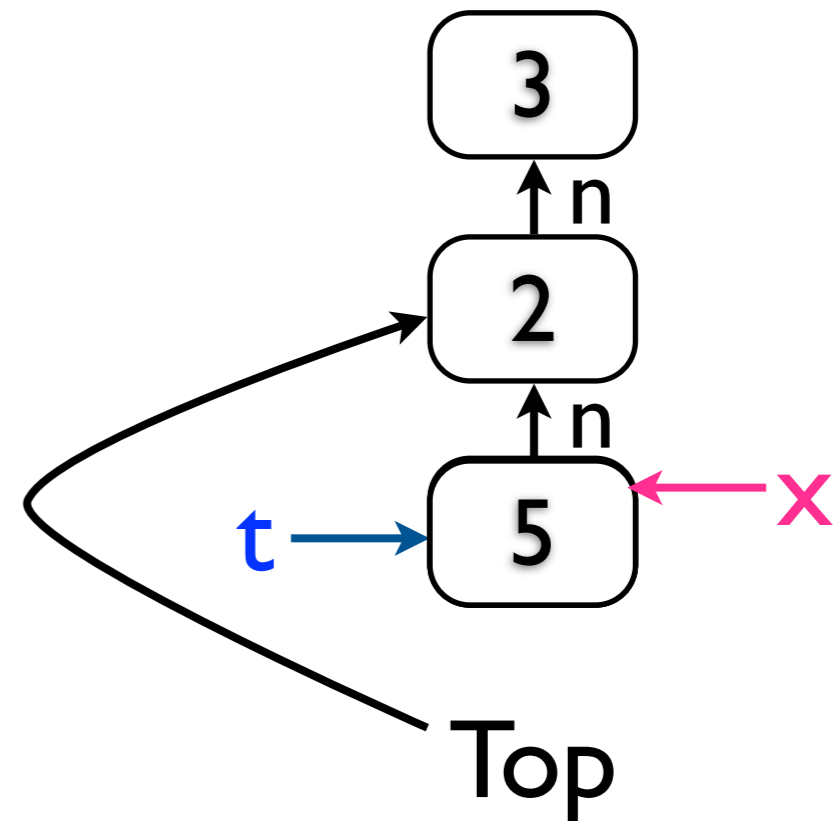
HP

--	--

$t = 0x..772$

“weird” races

```
pop() {  
  L1: t=Top  
push(k) {  
  x = new Node(k)  
  L2: t=Top  
  x.n = t  
  if (!CAS(&Top, t, x))  
    goto L2  
}  
  goto L1  
retire(t)  
}
```



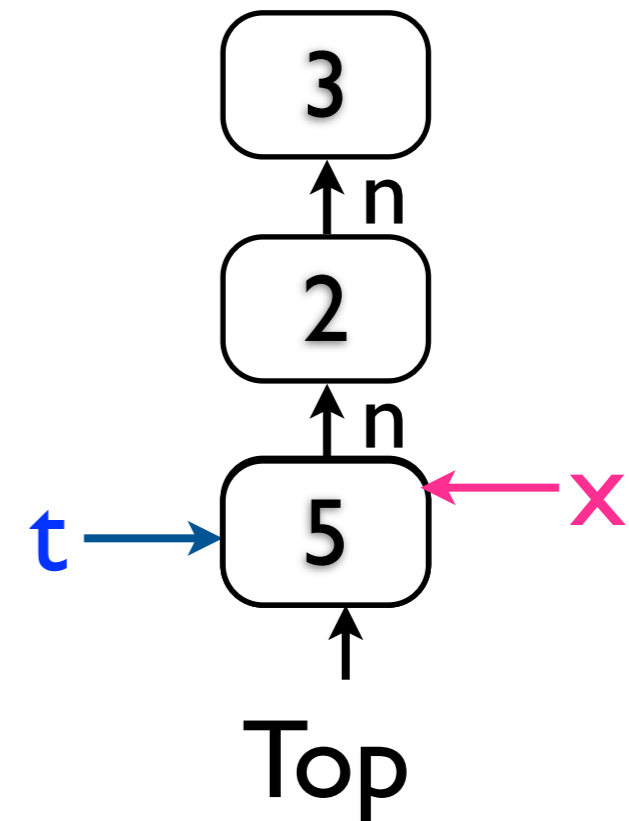
HP

--	--

$t = 0x..772$

“weird” races

```
pop() {  
  L1: t=Top  
push(k) {  
  x = new Node(k)  
  L2: t=Top  
  x.n = t  
  if (!CAS(&Top, t, x))  
    goto L2  
}  
  goto L1  
retire(t)  
}
```



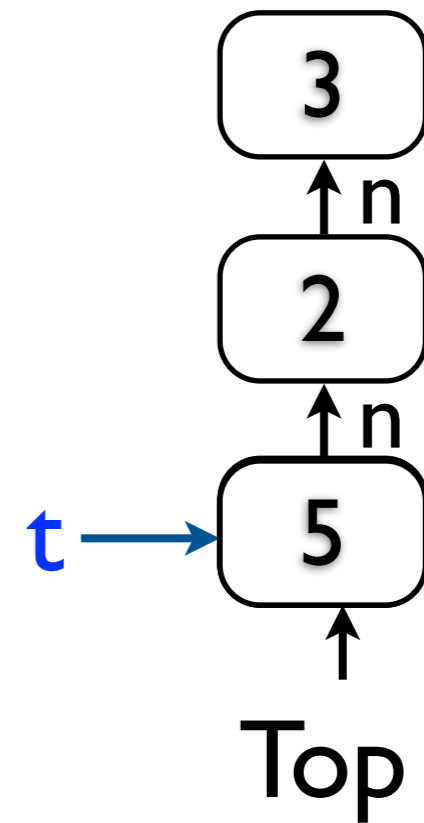
HP

--	--

$t = 0x..772$

“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



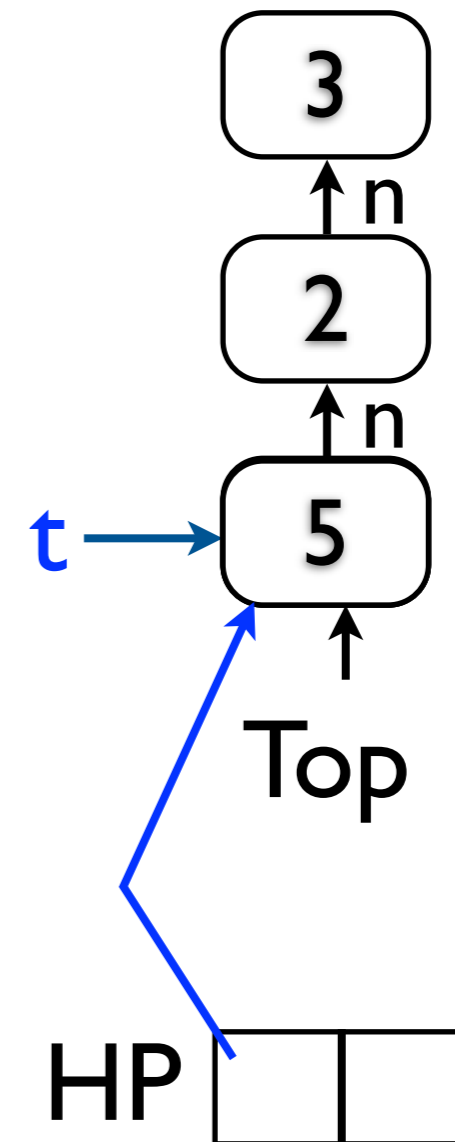
HP

--	--

$t = 0x..772$

“weird” races

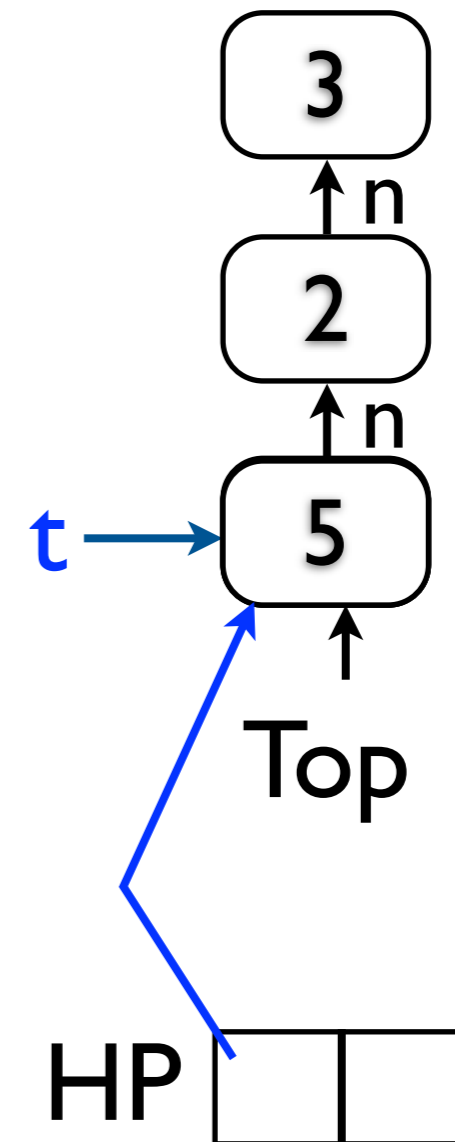
```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



$t = 0x...772$

“weird” races

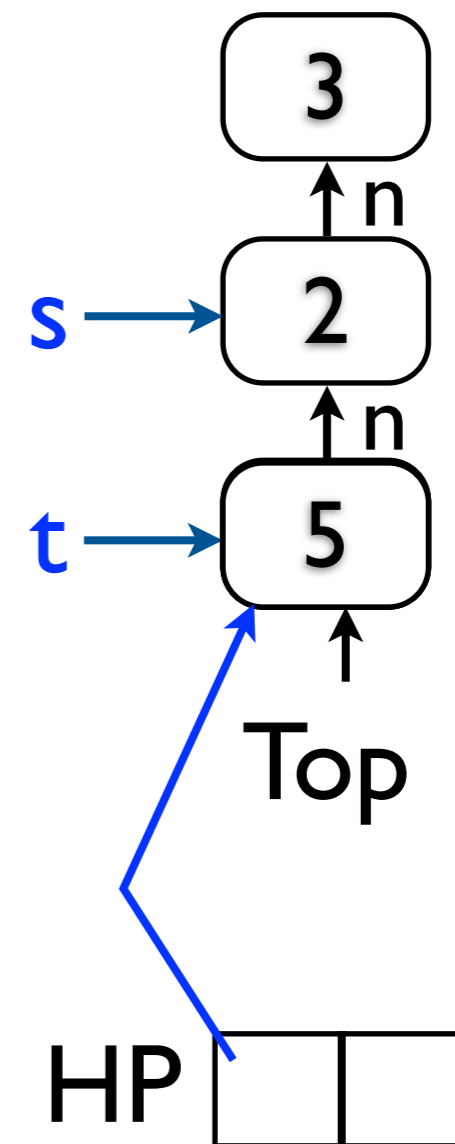
```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



$t = 0x...772$

“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



$t = 0x..772$

“weird” races

```
pop() {  
  L1: t=Top
```

benign use of a
dangling pointer

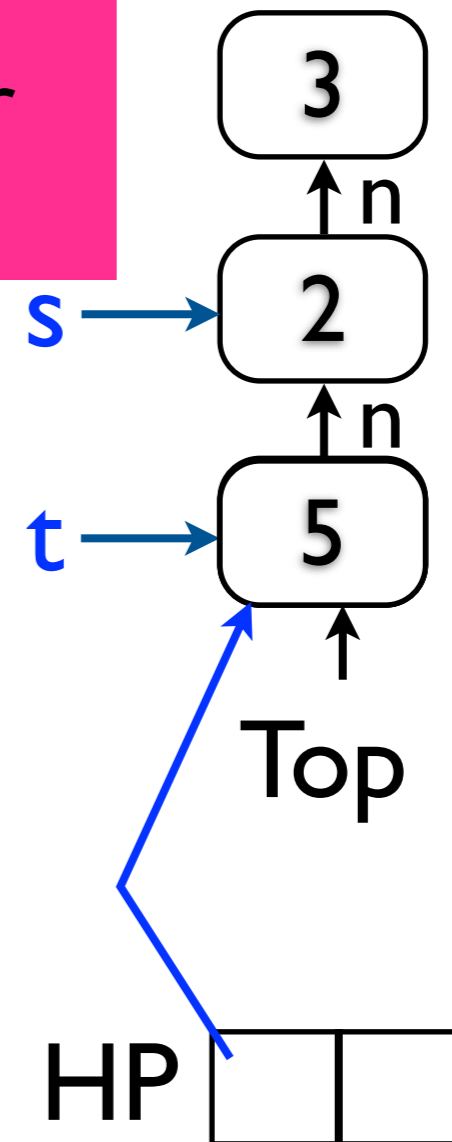
```
  HP[tid] = t
```

```
  if (t != Top) goto L1
```

```
  s=t.n
```

```
  if (!CAS(&Top, t, s))  
    goto L1
```

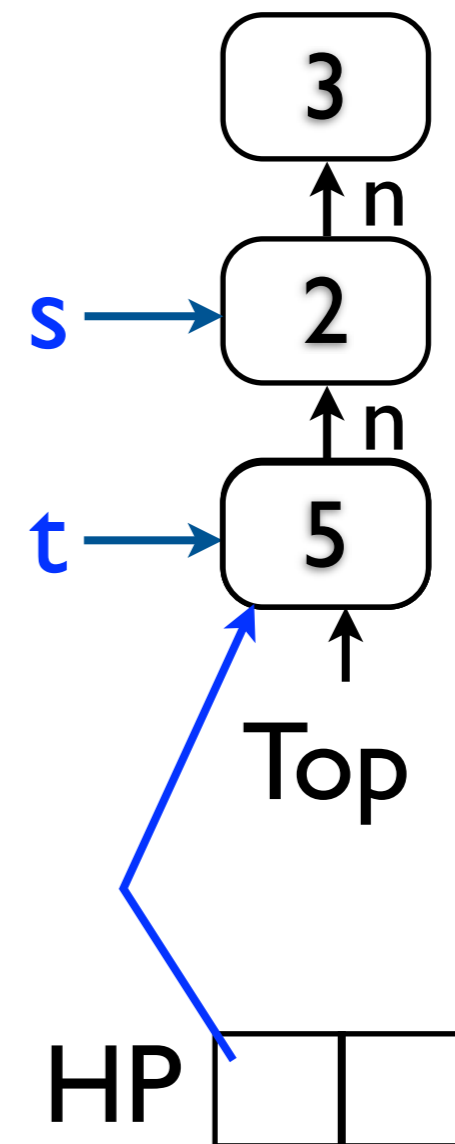
```
  retire(t)  
}
```



$t = 0x..772$

“weird” races

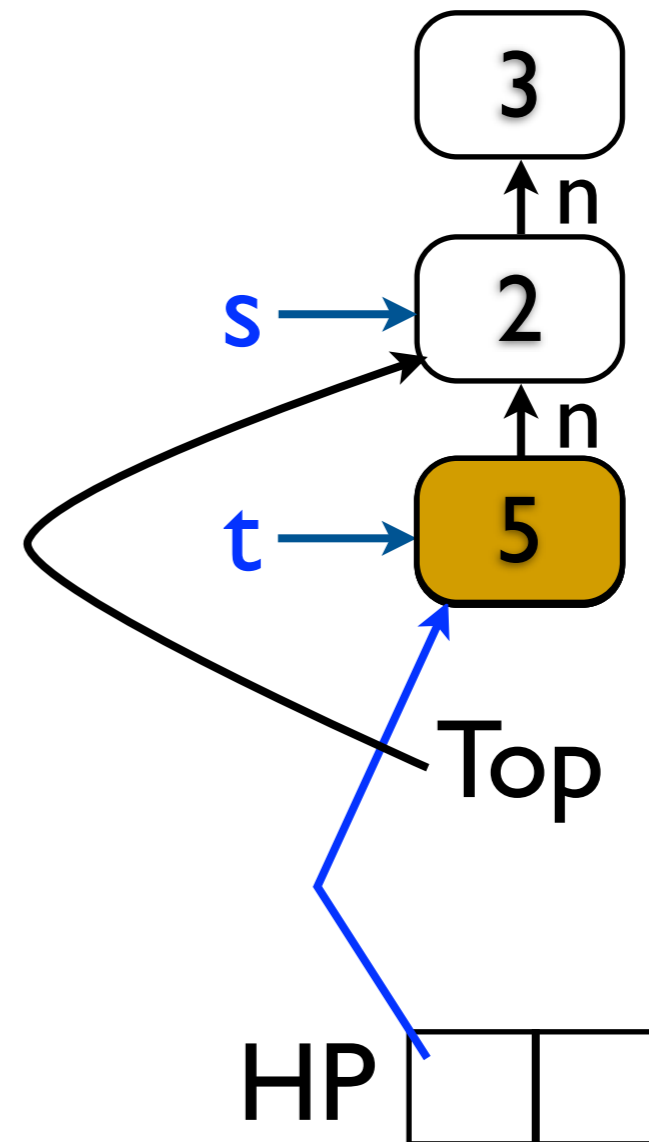
```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



$t = 0x..772$

“weird” races

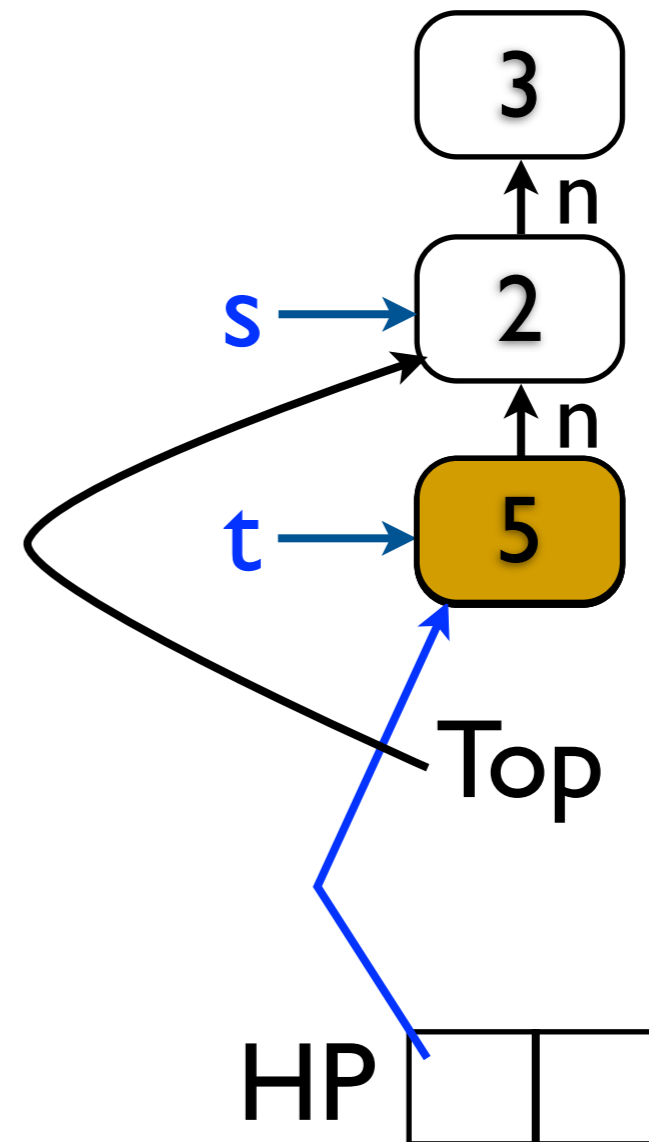
```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



$t = 0x...772$

“weird” races

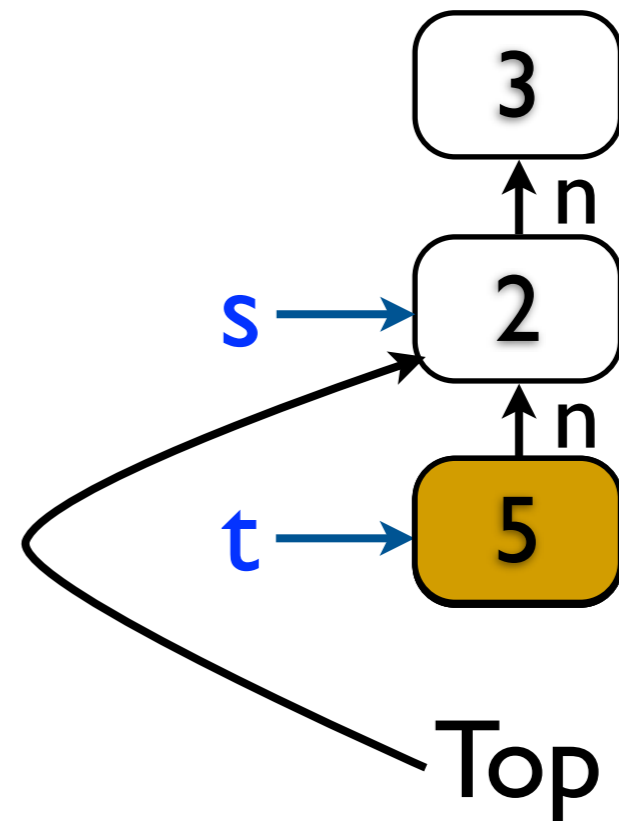
```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



$t = 0x...772$

“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



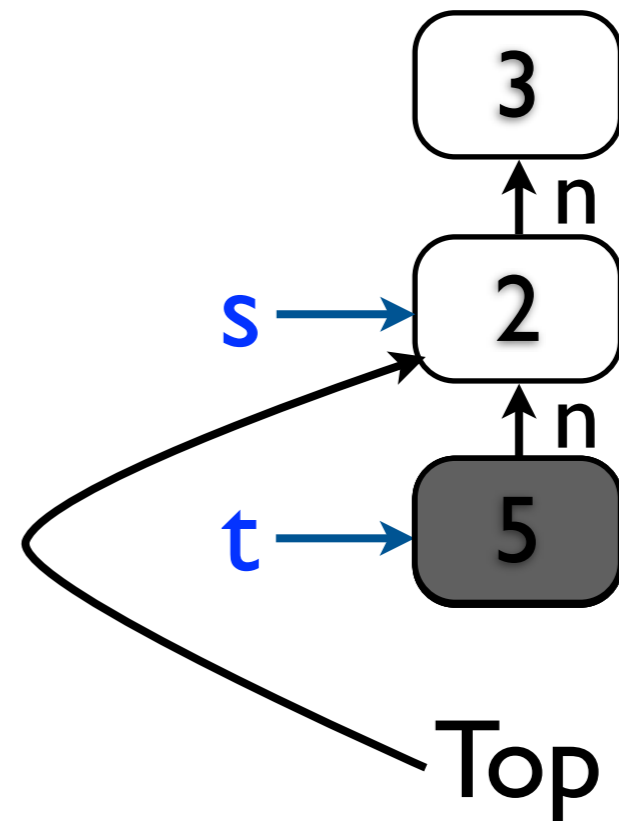
HP

--	--

$t = 0x..772$

“weird” races

```
pop() {  
  L1: t=Top  
  
  HP[tid] = t  
  
  if (t != Top) goto L1  
  
  s=t.n  
  
  if (!CAS(&Top, t, s))  
    goto L1  
  
  retire(t)  
}
```



HP

--	--

$t = 0x..772$

verification challenge

- verifying memory safety
 - shared memory concurrency
 - lock free data structures
 - no GC

our solution

- idiom-based verification using temporal Separation Logic

agenda

- verification problem: memory safety of highly concurrent data structures
- temporal idioms in verification problem
- temporal separation logic
 - Hoare Logic
 - Rely/Guarantee reasoning
 - Separation Logic
 - RGSep

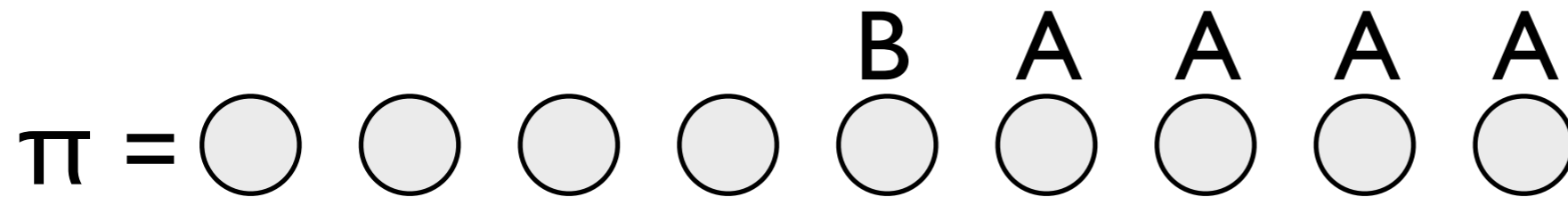
idiom: grace period

- **RCU** [McKenney Slingwine'98]
- **Hazard Pointers** [Michael'02]
- **Pass the Buck** [Herlihy Luchangco Moir'02]
- **Epoch** [Fraser Haris'03]
- programming methodologies
- per data structure specialization

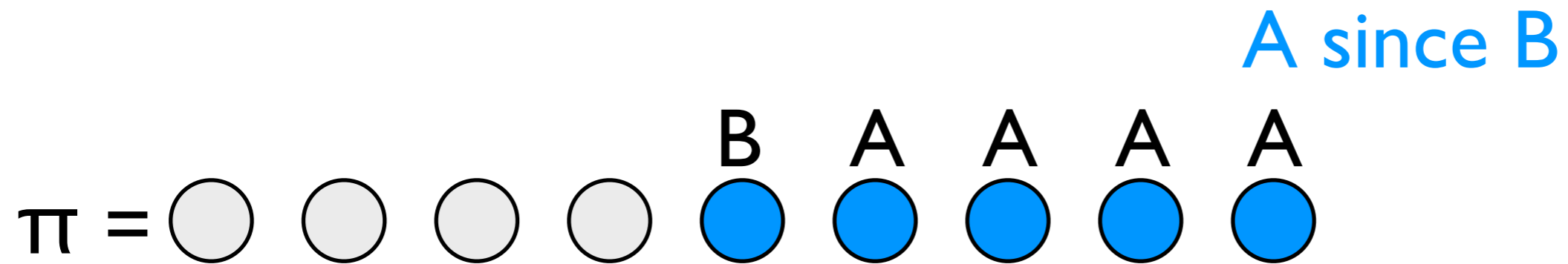
idiom: grace period

- pattern for temporal invariant
 - $A \text{ since } B \Rightarrow C$
- idiomatic use of a temporal tautology
 - $\neg B \text{ since } \neg A \Rightarrow \neg(A \text{ since } B)$

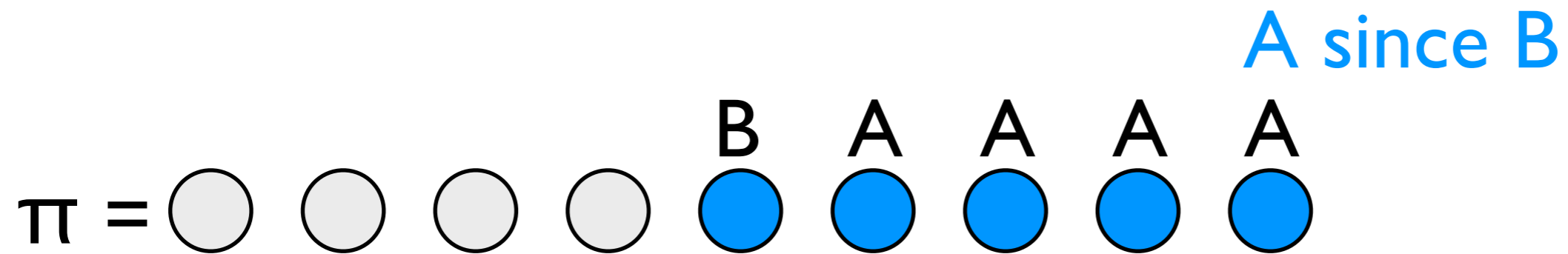
$\pi \models A$ since B



$\pi \models A \text{ since } B$



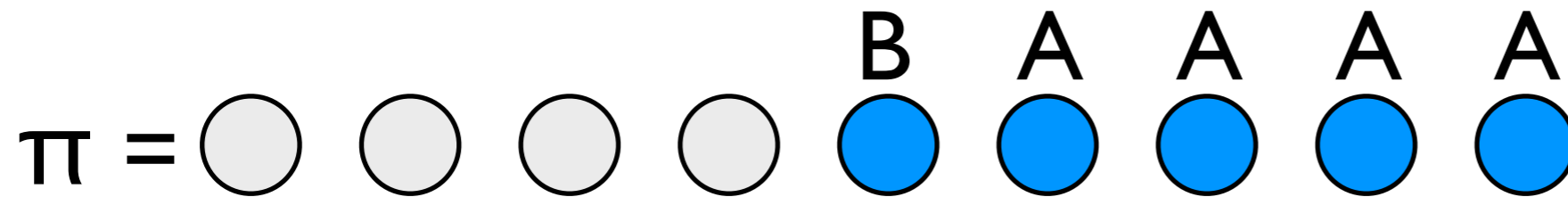
$$\pi \models A \text{ since } B \Rightarrow C$$



$\pi \models A \text{ since } B \Rightarrow C$

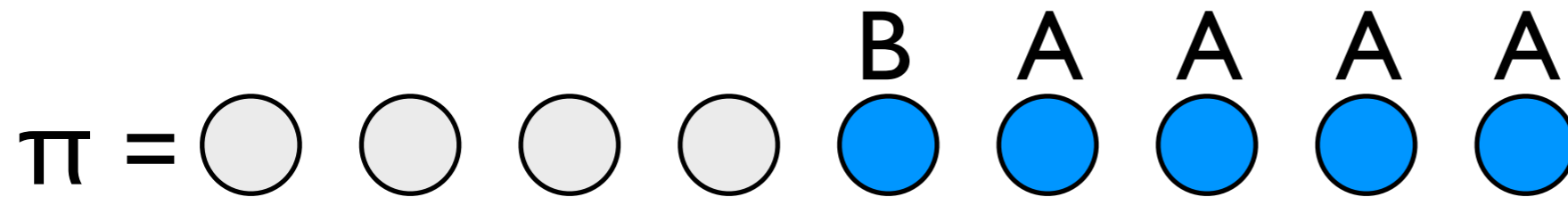
$\Rightarrow C$

$A \text{ since } B$



$$\pi \models A \text{ since } B \Rightarrow C$$

$$\Rightarrow C$$

$$A \text{ since } B$$


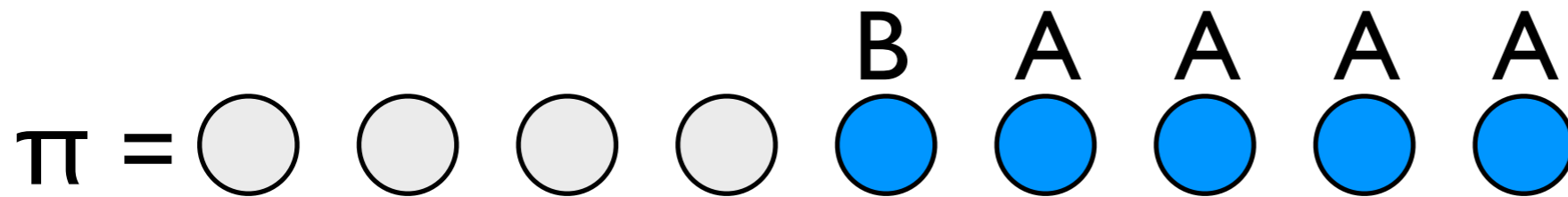
$$\forall \text{Tid } i \quad \forall \text{ location } x \quad \mathbf{A}_{i,x}: \text{HP}[i] = x$$

$$\mathbf{B}_x: \text{Top} = x$$

$$\mathbf{C}_x: x \text{ is allocated}$$

$$\pi \models A \text{ since } B \Rightarrow C$$

$$\Rightarrow C$$

$$A \text{ since } B$$


$$\forall \text{Tid } i \quad \forall \text{ location } x \quad \mathbf{A}_{i,x}: \text{HP}[i] = x$$

$$\mathbf{B}_x: \text{Top} = x$$

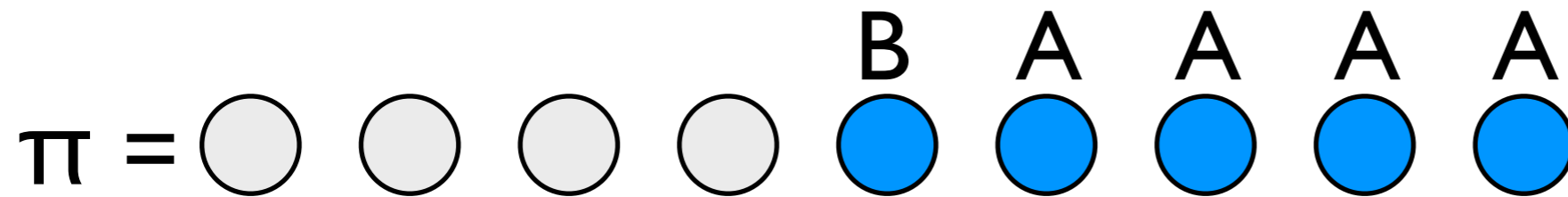
$$\mathbf{C}_x: x \text{ is allocated}$$

if a hazard pointer points to an object continuously
since the object is “safe” (in the data structure) then
the object is not reclaimed

$\pi \models A \text{ since } B \Rightarrow C$

$\Rightarrow C$

$A \text{ since } B \wedge A$

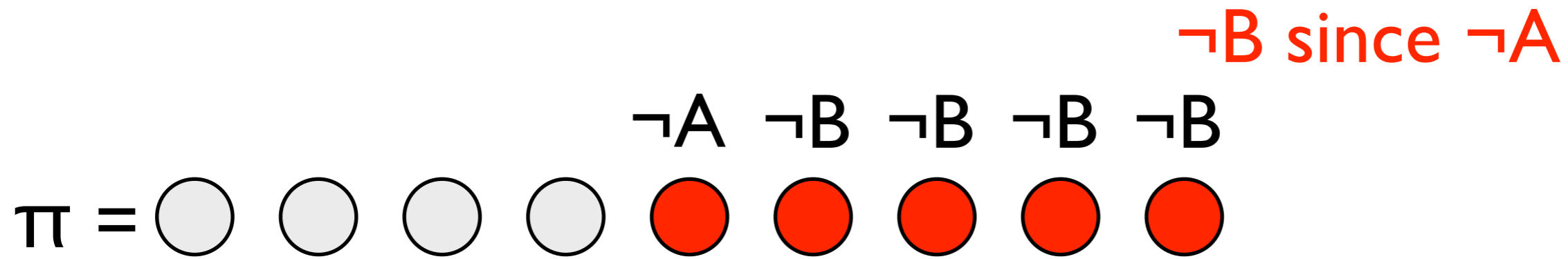


$B \quad A \quad A \quad A \quad A$

$\forall \text{Tid } i \quad \forall \text{ location } x \quad \mathbf{A}_{i,x}: \text{HP}[i] = x$

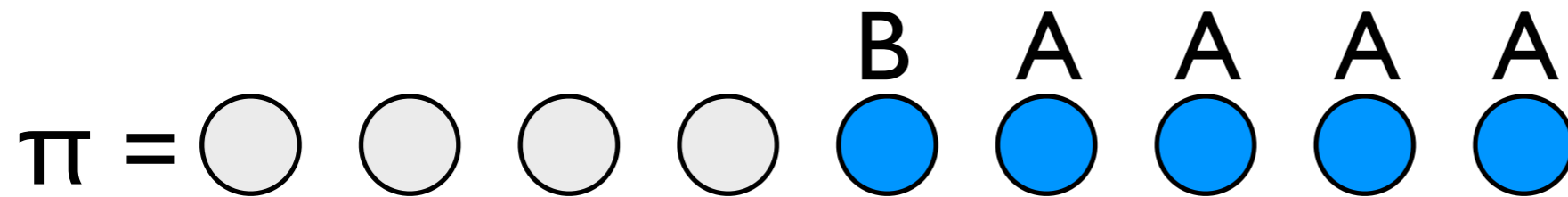
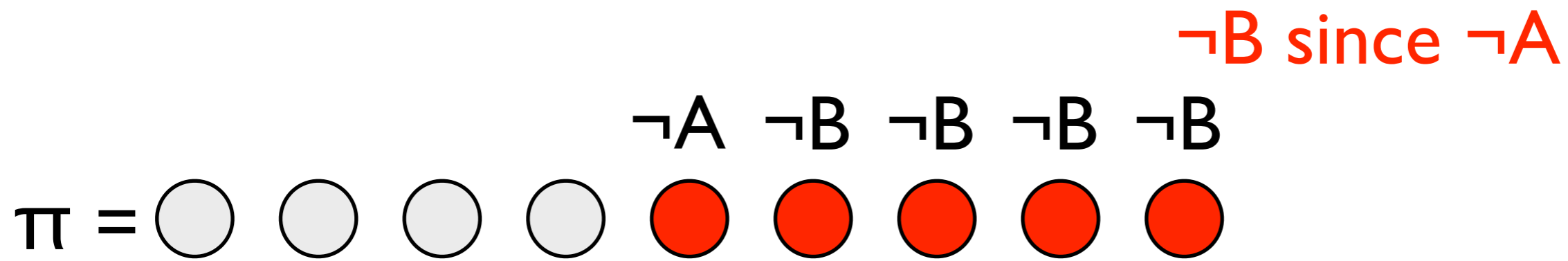
$\mathbf{B}_x: \text{Top} = x$

$\mathbf{C}_x: x \text{ is allocated}$



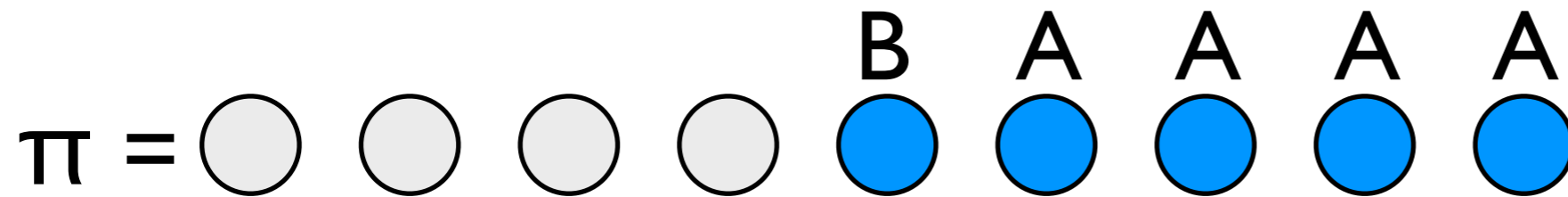
$\neg A \quad \neg B \quad \neg B \quad \neg B \quad \neg B$

$$\pi \models A \text{ since } B \Rightarrow C$$

 $\Rightarrow C$
 $A \text{ since } B \wedge A$

 $\forall \text{Tid } i \quad \forall \text{ location } x \quad \mathbf{A}_{i,x}: \text{HP}[i] = x$
 $\mathbf{B}_x: \text{Top} = x$
 $\mathbf{C}_x: x \text{ is allocated}$


$$\neg B \text{ since } \neg A \Rightarrow \neg(A \text{ since } B)$$

$$\pi \models A \text{ since } B \Rightarrow C$$

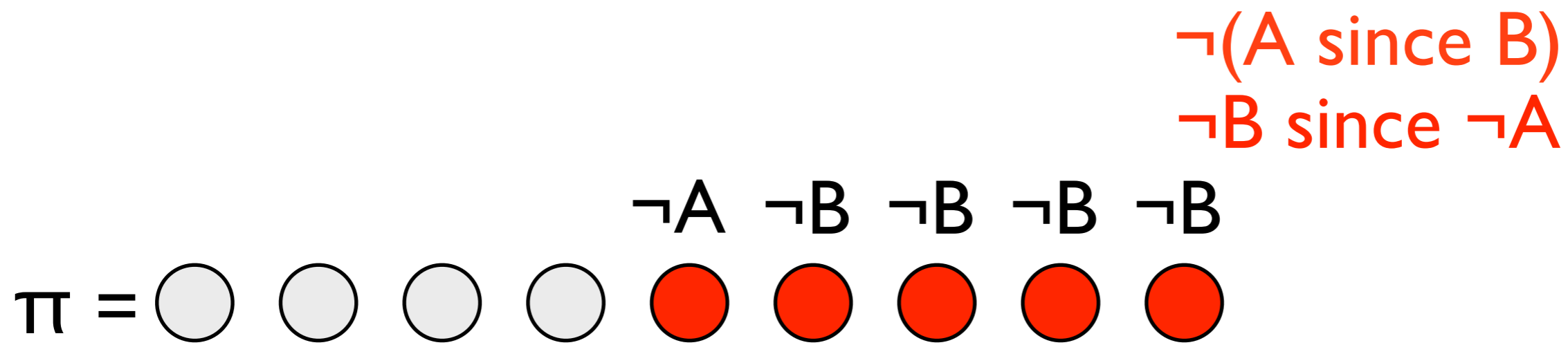
 $\Rightarrow C$
 $A \text{ since } B \wedge A$


$B \quad A \quad A \quad A \quad A$

$\forall \text{Tid } i \quad \forall \text{ location } x \quad \mathbf{A}_{i,x}: \text{HP}[i] = x$

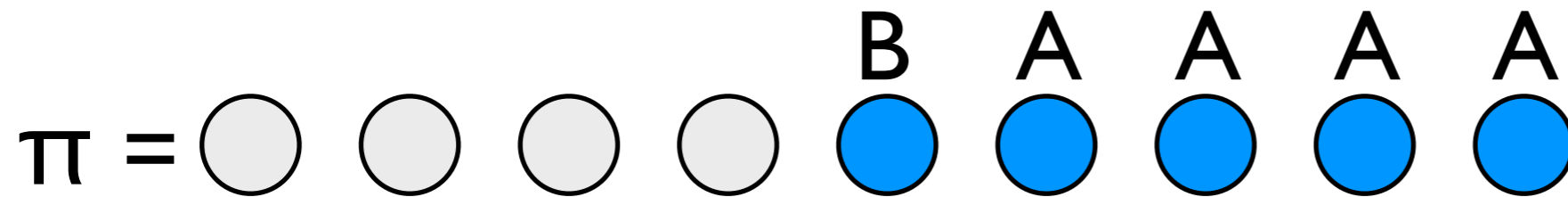
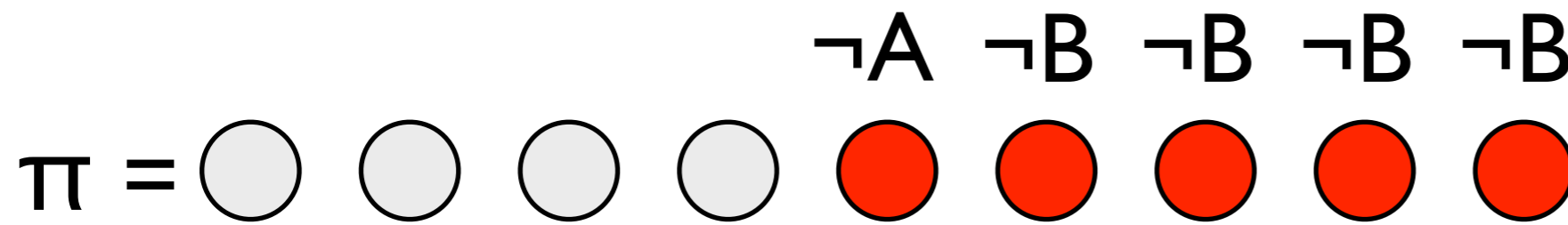
$\mathbf{B}_x: \text{Top} = x$

$\mathbf{C}_x: x \text{ is allocated}$

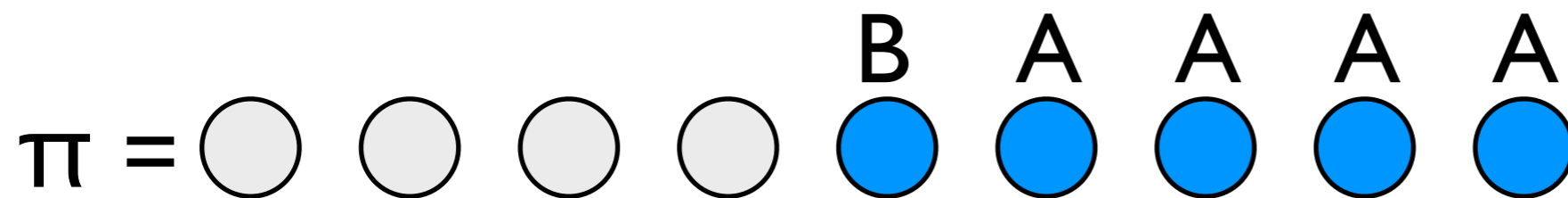


$\neg B \text{ since } \neg A \Rightarrow \neg(A \text{ since } B)$

$$\pi \models A \text{ since } B \Rightarrow C$$

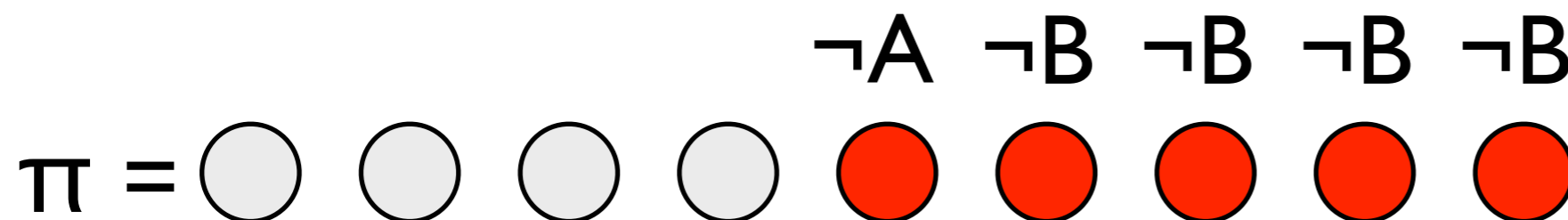
 $\Rightarrow C$
 $A \text{ since } B \wedge A$

 $\forall \text{Tid } i \quad \forall \text{ location } x \quad \mathbf{A}_{i,x}: \text{HP}[i] = x$
 $\mathbf{B}_x: \text{Top} = x$
 $\mathbf{C}_x: x \text{ is allocated}$
 $C?$
 $\neg(A \text{ since } B)$
 $\neg B \text{ since } \neg A$

 $\neg B \text{ since } \neg A \Rightarrow \neg(A \text{ since } B)$

$$\pi \models A \text{ since } B \Rightarrow C$$

 $\Rightarrow C$
 $A \text{ since } B \wedge A$

 $\forall \text{Tid } i \quad \forall \text{ location } x \quad \mathbf{A}_{i,x}: \text{HP}[i] = x$
 $\mathbf{B}_x: \text{Top} = x$
 $\mathbf{C}_x: x \text{ is allocated}$

the invariant holds
regardless of C

C?

 $\neg(A \text{ since } B)$
 $\neg B \text{ since } \neg A$

 $\neg B \text{ since } \neg A \Rightarrow \neg(A \text{ since } B)$

$\pi \models A \text{ since } B \Rightarrow C$

```
pop() {  
  L1: t=Top  
  HP[tid] = t
```

$\forall \text{Tid } i \ \forall \text{ location } x \ \mathbf{A}_{i,x}: \text{HP}[i] = x$

$\mathbf{B}_x: \text{Top} = x$

$\mathbf{C}_x: x \text{ is allocated}$

```
  if (t != Top) goto L1
```

```
  s=t.n
```

```
  if (!CAS(&Top, t, s))  
    goto L1
```

```
  retire(t)  
}
```

$\pi \models A \text{ since } B \Rightarrow C$

```
pop() {  
  L1: t=Top
```

```
  HP[tid] = t
```

```
  { Atid,t: HP[tid]=t }
```

```
  if (t != Top) goto L1
```

```
s=t.n
```

```
if (!CAS(&Top, t, s))  
  goto L1
```

```
retire(t)  
}
```

$\forall \text{Tid } i \ \forall \text{ location } x \ \mathbf{A}_{i,x}: \text{HP}[i] = x$

$\mathbf{B}_x: \text{Top} = x$

$\mathbf{C}_x: x \text{ is allocated}$

$\pi \models A \text{ since } B \Rightarrow C$

```
pop() {  
  L1: t=Top
```

```
  HP[tid] = t
```

```
  { Atid,t: HP[tid]=t }
```

```
  if (t != Top) goto L1
```

$\forall \text{Tid } i \ \forall \text{ location } x \ \mathbf{A}_{i,x}: \text{HP}[i] = x$

$\mathbf{B}_x: \text{Top} = x$

$\mathbf{C}_x: x \text{ is allocated}$

Top=t \wedge HP[tid]=t

```
s=t.n
```

```
if (!CAS(&Top, t, s))  
  goto L1
```

```
retire(t)
```

```
}
```

$\pi \models A \text{ since } B \Rightarrow C$

```
pop() {  
  L1: t=Top
```

```
  HP[tid] = t
```

```
  { Atid,t: HP[tid]=t }
```

```
  if (t != Top) goto L1
```

```
  { Atid,t: since Bt: HP[tid]=t since Top=t  $\wedge$  HP[tid]=t }
```

```
  s=t.n
```

```
  if ( !CAS(&Top, t, s) )  
    goto L1
```

```
  retire(t)  
}
```

$\forall \text{Tid } i \ \forall \text{ location } x \ \mathbf{A}_{i,x}: \text{HP}[i] = x$

$\mathbf{B}_x: \text{Top} = x$

$\mathbf{C}_x: x \text{ is allocated}$

Top=t \wedge HP[tid]=t

$\pi \models A \text{ since } B \Rightarrow C$

```
pop() {  
  L1: t=Top
```

```
  HP[tid] = t
```

```
  { Atid,t: HP[tid]=t }
```

```
  if (t != Top) goto L1
```

```
  { Atid,t: since Bt: HP[tid]=t since Top=t  $\wedge$  HP[tid]=t }
```

```
  s=t.n
```

```
  if ( !CAS(&Top, t, s) )  
    goto L1
```

```
  retire(t)  
}
```

$\forall \text{Tid } i \ \forall \text{ location } x \ \mathbf{A}_{i,x}: \text{HP}[i] = x$

$\mathbf{B}_x: \text{Top} = x$

$\mathbf{C}_x: x \text{ is allocated}$

Top=t \wedge HP[tid]=t

\Rightarrow

{ Ct: t is allocated }

$\pi \models A \text{ since } B \Rightarrow C$

```
retire(N r) {  
     $\forall \text{Tid } i \ \forall \text{ location } x$  Ai,x: HP[i] = x  
    Bx: Top = x  
    Cx: x is allocated  
  
    i = 0;  
    while(i < NTID) {  
        while (HP[i] == r) skip  
        i++  
    }  
  
    free (r)  
}
```

$\pi \models A \text{ since } B \Rightarrow C$

```
retire(N r) {
```

```
  {  $\neg B_t: \text{Top} \neq r$  }
```

```
  i = 0;
```

```
  while(i < NTID) {
```

```
    while (HP[i] == r) skip
```

```
    i++
```

```
  }
```

```
  free (r)
```

```
}
```

$\forall \text{Tid } i \ \forall \text{ location } x \ \mathbf{A}_{i,x}: \text{HP}[i] = x$

$\mathbf{B}_x: \text{Top} = x$

$\mathbf{C}_x: x \text{ is allocated}$

$\pi \models A \text{ since } B \Rightarrow C$

```
retire(N r) {  
  {  $\neg B_t: \text{Top} \neq r$  }
```

$\forall \text{Tid } i \ \forall \text{ location } x \ \mathbf{A}_{i,x}: \text{HP}[i] = x$

$\mathbf{B}_x: \text{Top} = x$

$\mathbf{C}_x: x \text{ is allocated}$

```
  i = 0;
```

```
  while(i < NTID) {
```

$\text{Top} \neq r \wedge \text{HP}[i] \neq r$

```
    while (HP[i] == r) skip
```

```
    i++
```

```
  }
```

```
  free (r)
```

```
}
```

$\pi \models A \text{ since } B \Rightarrow C$

```
retire(N r) {
```

```
  {  $\neg B_t: \text{Top} \neq r$  }
```

```
  i = 0;
```

```
  while(i < NTID) {
```

```
    while (HP[i] == r) skip
```

```
    i++
```

```
  }
```

```
  {  $\forall i \leq \text{NTID}. \neg B_r \text{ since } \neg A_{i,r}:$ 
```

```
     $\forall i \leq \text{NTID}. \text{Top} \neq r \text{ since } \text{Top} \neq r$  }
```

```
  free (r)
```

```
}
```

$\forall \text{Tid } i \ \forall \text{ location } x \ \mathbf{A}_{i,x}: \text{HP}[i] = x$

$\mathbf{B}_x: \text{Top} = x$

$\mathbf{C}_x: x \text{ is allocated}$

$\text{Top} \neq r \wedge \text{HP}[i] \neq r$

$\pi \models A \text{ since } B \Rightarrow C$

```
retire(N r) {
```

```
  {  $\neg B_t: \text{Top} \neq r$  }
```

```
  i = 0;
```

```
  while(i < NTID) {
```

```
    while (HP[i] == r) skip
```

```
    i++
```

```
  }
```

```
  {  $\forall i \leq \text{NTID}. \neg B_r \text{ since } \neg A_{i,r}:$ 
```

```
     $\forall i \leq \text{NTID}. \text{Top} \neq r \text{ since } \text{Top} \neq r$  }
```

```
  free (r)
```

```
}
```

$\forall \text{Tid } i \ \forall \text{ location } x \ \mathbf{A}_{i,x}: \text{HP}[i] = x$

$\mathbf{B}_x: \text{Top} = x$

$\mathbf{C}_x: x \text{ is allocated}$

$\text{Top} \neq r \wedge \text{HP}[i] \neq r$

$\neg B \text{ since } \neg A \Rightarrow \neg(A \text{ since } B)$

$\pi \models A \text{ since } B \Rightarrow C$

```
retire(N r) {  
  {  $\neg B_t$ :  $\text{Top} \neq r$  }
```

$\forall \text{Tid } i \ \forall \text{ location } x \ \mathbf{A}_{i,x}: \text{HP}[i] = x$

$\mathbf{B}_x: \text{Top} = x$

$\mathbf{C}_x: x \text{ is allocated}$

```
  i = 0;
```

```
  while(i < NTID) {
```

$\text{Top} \neq r \wedge \text{HP}[i] \neq r$

```
    while (HP[i] == r) skip
```

```
    i++
```

$\neg B \text{ since } \neg A \Rightarrow \neg(A \text{ since } B)$

```
  }
```

```
{  $\forall i \leq \text{NTID}. \neg B_r \text{ since } \neg A_{i,r}$ :
```

$\forall i \leq \text{NTID}. \text{Top} \neq r \text{ since } \text{Top} \neq r$

\Rightarrow

```
  free (r)
```

$\forall i \leq \text{NTID}. \neg(\text{HP}[i] = r \text{ since } \text{Top} = r)$

```
}
```

$\pi \models A \text{ since } B \Rightarrow C$

```
retire(N r) {  
  {  $\neg B_t$ :  $\text{Top} \neq r$  }
```

$\forall \text{Tid } i \ \forall \text{ location } x \ \mathbf{A}_{i,x}: \text{HP}[i] = x$

$\mathbf{B}_x: \text{Top} = x$

$\mathbf{C}_x: x \text{ is allocated}$

```
  i = 0;
```

```
  while(i < NTID) {
```

$\text{Top} \neq r \wedge \text{HP}[i] \neq r$

```
    while (HP[i] == r) skip
```

```
    i++
```

$\neg B \text{ since } \neg A \Rightarrow \neg(A \text{ since } B)$

```
  }
```

```
{  $\forall i \leq \text{NTID}. \neg B_r \text{ since } \neg A_{i,r}$ :
```

$\forall i \leq \text{NTID}. \text{Top} \neq r \text{ since } \text{Top} \neq r$

\Rightarrow

```
  free (r)
```

$\forall i \leq \text{NTID}. \neg(\text{HP}[i] = r \text{ since } \text{Top} = r)$

```
}
```

// r can be deallocated

agenda

- verification problem: memory safety of highly concurrent data structures
- temporal idioms in verification problem
- temporal separation logic
 - Hoare Logic
 - Rely/Guarantee reasoning
 - Separation Logic
 - RGSep

Hoare logics

- programming language
- assertion language
- Hoare triples
- proof rules

programming language

- $P = C \parallel \dots \parallel C$
- $C = c \mid C; C \mid C+C \mid C^*$
- $c = x := n \mid x := y + z \mid \text{assert } (x = y) \mid \dots$
- no heap

small step operational semantics

- $\sigma \in \Sigma = \text{Var} \rightarrow \text{Int}$
- $\sigma = [x \mapsto 9, y \mapsto 3]$

$$\langle C_t, \sigma \rangle \leadsto \langle C'_t, \sigma' \rangle$$

$$\langle C_1 \mid \mid \dots \mid \mid C_t \mid \mid \dots \mid \mid C_n, \sigma \rangle \rightarrow \langle C_1 \mid \mid \dots \mid \mid C'_t \mid \mid \dots \mid \mid C_n, \sigma' \rangle$$

assertion language

- $A = p \mid p \wedge p \mid p \vee p \mid \exists x. p(x) \mid \dots$
- $p = x = n \mid x > n \mid \dots$
- $[[p]] = 2^\Sigma$
- $[[x = 3 \wedge y > 4]] = \{\sigma \in \Sigma \mid \sigma(x)=3 \wedge \sigma(y) > 4\}$

Hoare triples

- $\{P\} C \{Q\}$
 - P, Q are assertions
 - C command
- $\{P\} C \{Q\} = \forall \sigma \in [[P]] \text{ if } (C, \sigma) \rightarrow^* \sigma \text{ then } \sigma \in [[Q]]$

proof rules

$$\frac{\begin{array}{c} \{P_1\} C_1 \{P_2\} \\ \{P_2\} C_2 \{P_3\} \end{array}}{\{P_1\} C_1; C_2 \{P_3\}} \text{SEQ}$$

$$\frac{\begin{array}{c} \{P\} C_1 \{Q\} \\ \{P\} C_2 \{Q\} \end{array}}{\{P\} C_1 + C_2 \{Q\}} \text{CHOICE}$$

$$\frac{\{P\} C \{P\}}{\{P\} C^* \{P\}} \text{LOOP}$$

$$\frac{\begin{array}{c} \{P_1\} C \{Q_1\} \\ \{P_2\} C \{Q_2\} \end{array}}{\{P_1 \vee P_2\} C \{Q_1 \vee Q_2\}} \text{DISJ}$$

proof rules & separation

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

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

proof rules & separation

Rule of constancy **aliasing**

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

Disjoint parallelism rule

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

proof rules & separation

Rule of constancy **aliasing**

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

Disjoint parallelism rule **shared resources**

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

problem: interference

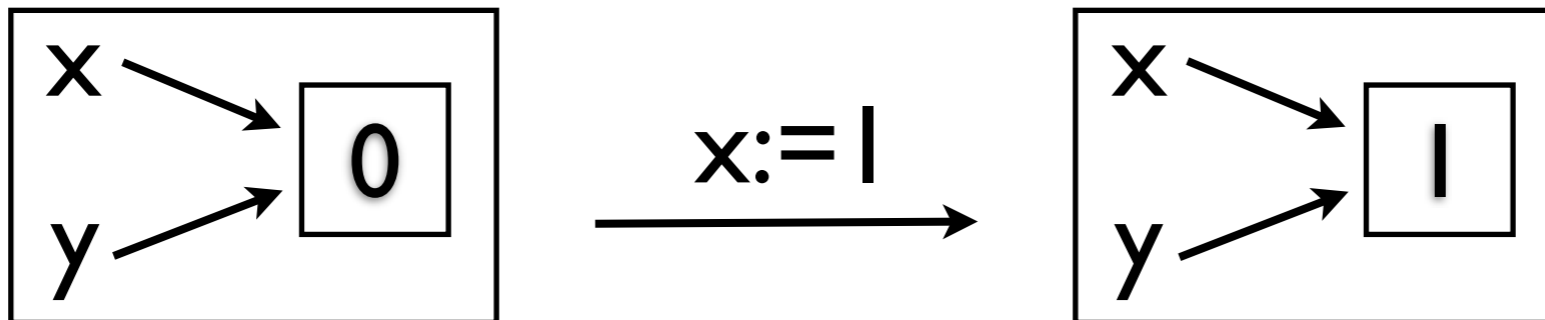
$$\frac{\{x = 0\} \ x := 1 \ \{x = 1\}}{\{x = 0 \wedge y = 0\} \ x := 1 \ \{x = 1 \wedge y = 0\}} \quad \text{fv}(y = 0) \cap \text{fv}(x := 1) = \emptyset$$

problem: interference

$$\{x = 0\} \ x := 1 \ \{x = 1\}$$

$$\{x = 0 \wedge y = 0\} \ x := 1 \ \{x = 1 \wedge y = 0\} \quad \text{fv}(y = 0) \cap \text{fv}(x := 1) = \emptyset$$

aliasing

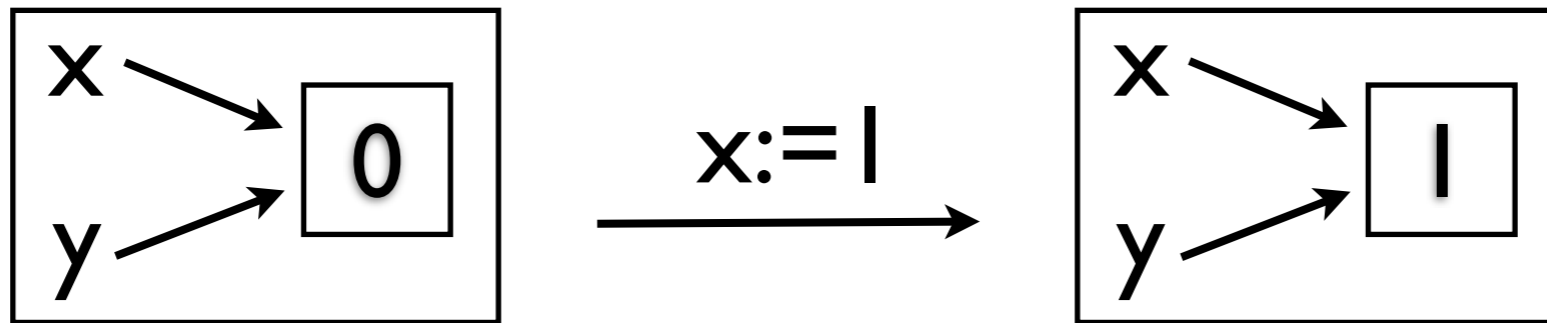


problem: interference

$$\{x = 0\} \ x:=1 \ \{x=1\}$$

$$\{x = 0 \wedge y = 0\} \ x:=1 \ \{x=1 \wedge y = 0\} \quad \text{fv}(y = 0) \cap \text{fv}(x:=1) = \emptyset$$

aliasing



$$\{true\} \ x:=1 \ \{x=1\} \quad \{true\} \ x:=1 \ \{x=1\}$$

$$\{true\} \ x:=1 \parallel x:=1 \ \{?\}$$

shared resources

$$\text{fv}(x:=1) \cap$$

$$\text{fv}(x:=1) \neq \emptyset$$

Hoare logics + R/G

- programming language
- assertion language

- Hoare triples
- proof rules

Hoare triples

- $R, G \vdash \{P\} C \{Q\}$
 - R, G describe interference: $[[R]] = [[G]] \in \Sigma \times \Sigma$
- $R, G \vdash \{P\} C \{Q\} = \forall \sigma \in [[P]]$ if every transition of C is in $[[R]]$ and every transition of the environment is in $[[G]]$ and $(C, \sigma) \rightarrow^* \sigma$ then $\sigma \in [[Q]]$

proof rules

$$\frac{\begin{array}{l} R \cup G', G \vdash \{P\} C \{Q\} \\ R \cup G, G' \vdash \{P\} C' \{Q\} \end{array}}{R, G \cup G' \vdash \{P\} C \parallel C' \{Q\}}$$

$$G = G' = \{(x = n) \leadsto (x = l) \mid n \in \text{Nat}\}$$

$$\emptyset, G \vdash \{\text{true}\} x := l \{x = l\}$$

$$\emptyset, G' \vdash \{\text{true}\} x := l \{x = l\}$$

$$\emptyset, G \cup G' \vdash \{\text{true} \wedge \text{true}\} x := l \parallel x := l \{x = l \wedge x = l\}$$

stability

- $R, G \vdash \{P\} C \{Q\}$
 - R, G describe interference: $[[R]] = [[G]] \in \Sigma \times \Sigma$
 - P, Q are **stable** under R :
 - $\forall \sigma, \sigma'$ if $\sigma \in [[P]]$ and $(\sigma, \sigma') \in [[R]]$ then $\sigma' \in [[P]]$
- $R, G \vdash \{P\} C \{Q\} = \forall \sigma \in [[P]]$ if every transition of C is in $[[R]]$ and every transition of the environment is in $[[G]]$ and $(C, \sigma) \rightarrow^* \sigma$ then $\sigma \in [[Q]]$

separation logic

- programming language
- assertion language
- Hoare triples
- proof rules

separation logic

- programming language
- assertion language
- Hoare triples
- proof rules

- SL is a convenient logic to reason on heaps

programming language

- $P = C \parallel \dots \parallel C$
- $C = c \mid C;C \mid C+C \mid C^*$
- $c = x := n \mid x := y + z \mid \text{assert } (x = y) \mid \dots$
 $x := \text{new}() \mid [x] := y \mid x := [y] \mid \text{free}(x)$

programming language

- $P = C \parallel \dots \parallel C$
- $C = c \mid C;C \mid C+C \mid C^*$
- $c = x := n \mid x := y + z \mid \text{assert } (x = y) \mid \dots$
 $x := \text{new}() \mid [x] := y \mid x := [y] \mid \text{free}(x)$
- $\sigma \in \Sigma$ *substates*

RAM model

- $\Sigma = \text{Loc} \rightarrow \text{Val}$
 - $[7 \mapsto 9, 9 \mapsto 3]$
- $\sigma * \sigma' = \text{disjoint union}$
 - $[7 \mapsto 9, 9 \mapsto 3] * [10 \mapsto 9] = [7 \mapsto 9, 9 \mapsto 3, 10 \mapsto 9]$
 - $[7 \mapsto 9, 9 \mapsto 3] * [9 \mapsto 3] = \text{undefined}$
- $\text{emp} = []$

separation algebra

- $(\Sigma, *, e)$ - cancelative partial commutative monoid
 - Σ set
 - $\sigma * \sigma' = \sigma' * \sigma \in \Sigma$ or undefined
 - $\sigma * e = e * \sigma = \sigma$
 - $\sigma * \sigma' = \sigma * \sigma''$ *implies* $\sigma' = \sigma''$
- $\sigma \# \sigma'$ *iff* $\sigma * \sigma'$
- $\sigma \sqsubseteq \sigma'$ *iff* $\exists \sigma''. \sigma * \sigma'' = \sigma'$

local actions

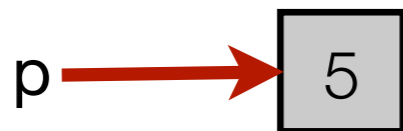
- $[[c]] : \Sigma \rightarrow 2^\Sigma \cup \{T\}$
 - $\sigma \sqsubseteq \sigma'$ iff $\exists \sigma''. \sigma * \sigma'' = \sigma' \vee \sigma' = T$
- $[[[7]:=3]] ([7 \mapsto 2]) = ([7 \mapsto 3])$
- $[[[7]:=3]] ([7 \mapsto 2] * [3 \mapsto 2]) = ([7 \mapsto 3], 3 \mapsto 2])$
- $[[[7]:=3]] ([8 \mapsto 2] * [3 \mapsto 2]) = T$

local actions

- $[[c]] : \Sigma \rightarrow 2^{\Sigma \cup \{T\}}$
 - $\sigma \sqsubseteq \sigma'$ iff $\exists \sigma''. \sigma * \sigma'' = \sigma' \vee \sigma' = T$
 - $\sigma \sqsubseteq \sigma'$: $[[c]](\sigma) \neq T$ implies $[[c]](\sigma') \neq T$ (safe monotonicity)
 - $[[c]](\sigma_0) \neq T$ and $\sigma' \in [[c]](\sigma_0 * \sigma_1)$ implies $\exists \sigma_0'. \sigma_0' \in [[c]](\sigma_0)$ and $\sigma' = \sigma_0' * \sigma_1$ (frame property)
 - $[[c]](\sigma * \sigma') \sqsubseteq [[c]](\sigma) * \sigma'$
 - $[[[7]:=3]] ([7 \mapsto 2] * [3 \mapsto 2]) \sqsubseteq [[[7]:=3]] ([7 \mapsto 2]) * [3 \mapsto 2]$

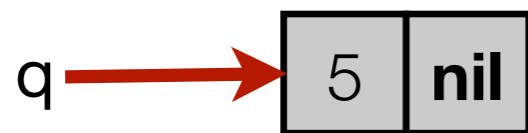
assertion language

- SL has a convenient syntax to describe heaps



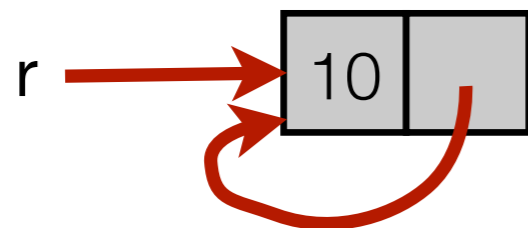
$p \mapsto 5$

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



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

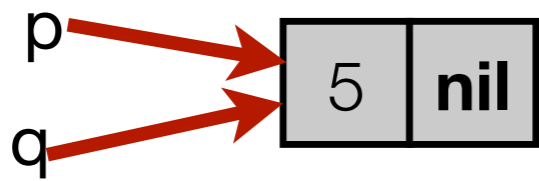
$\text{heap}(q) = 5 \wedge$
 $\text{heap}(q+1) = \text{nil}$



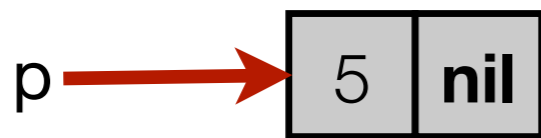
$r \mapsto 10, r$

assertion language

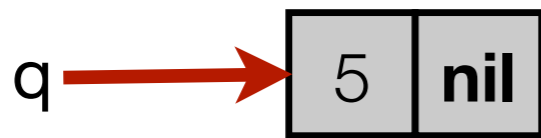
- SL has a convenient syntax to describe heaps



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



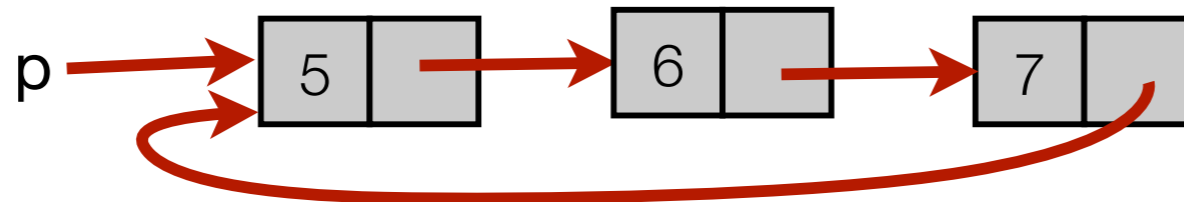
$$p \mapsto 5, \text{nil} \wedge q \mapsto 5, \text{nil} \wedge p \neq q$$



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

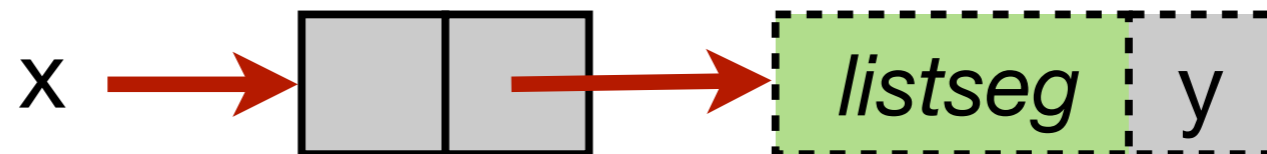
assertion language

- SL has a convenient syntax to describe heaps



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

$x=y$



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

Hoare triples

- $\{P\} C \{Q\}$
 - precondition must specify cells accessed by C
 - error avoiding meaning

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

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

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

frame rule

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

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

disjoint concurrency

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

C_1 and C_2 manipulate disjoint parts of the heap and share access to unmodified variables

RGSep

- programming language
- assertion language
- Hoare triples
- proof rules

RGSep

- programming language
- assertion language
- Hoare triples
- proof rules

- heaps + fine-grained concurrency

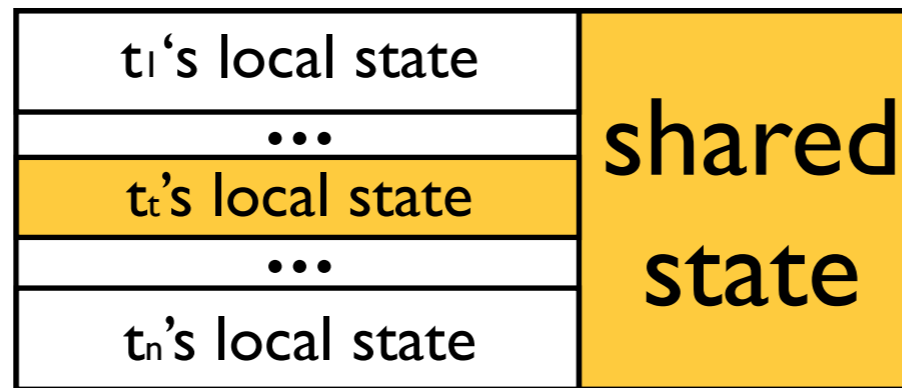
programming language

- $P = C \parallel \dots \parallel C$
- $C = c \mid C;C \mid C+C \mid C^* \mid \langle C \rangle$
- $c = x := n \mid x := y + z \mid \text{assert } (x = y) \mid \dots$
 $x := \text{new}() \mid [x] := y \mid x := [y] \mid \text{free}(x)$
- $\sigma \in \Sigma$ local *substate* + *global state*

global view

SL

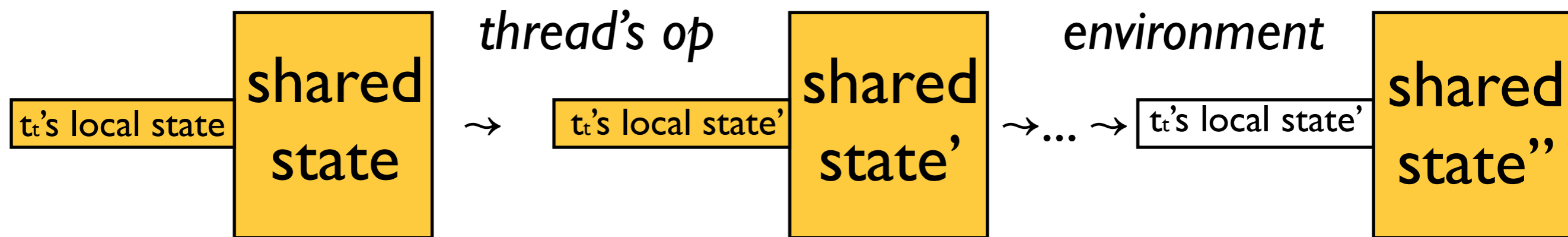
R/G



stack

heap

local view

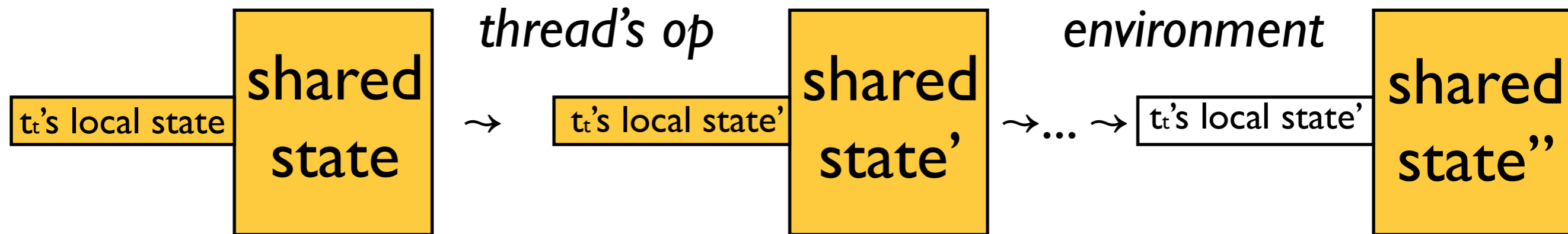


local view

$R_t, G_t \subseteq (\text{shared state}) \times (\text{shared state})$

assert $\bullet \leadsto \bullet \in G_t$

assume $\bullet \leadsto \bullet \in R_t$



assertion language

- P, Q, R - assertions in separation logic
- p, q, r - assertions in RGSep

local state

- $p = P \mid \boxed{P} \mid p * p \mid p \vee p \mid \forall x. p \mid \exists x. p$

shared state

- $[[p]] = 2^{\Sigma \times \Sigma}$

assertion language

- $p = P \mid \boxed{P} \mid p * p \mid p \vee p \mid \forall x. p \mid \exists x. p$
 - $(l, s) \models P \Leftrightarrow l \models P$
 - $(l, s) \models P \Leftrightarrow s \models P$
 - $(l, s) \models p * q \Leftrightarrow \exists l', l'' . l = l' * l'' \wedge (l', s) \models p \wedge (l'', s) \models q$
- $[[p]] = 2^{\Sigma \times \Sigma}$

proof rules

$$\frac{\begin{array}{l} R \cup G', G \vdash \{p\} C \{q'\} \\ R \cup G, G' \vdash \{p'\} C' \{q'\} \end{array}}{R, G \cup G' \vdash \{p * p'\} C \parallel C' \{q * q'\}}$$

$$(l, s) \models p * q \Leftrightarrow \exists l', l'' . l = l' * l'' \wedge \\ (l', s) \models p \wedge (l'', s) \models q$$

proof rules

$$\frac{\begin{array}{c} (P', Q') \in G \\ \{P * P'\} \langle C \rangle \{Q * Q'\} \end{array}}{R, G \vdash \{P * \boxed{P'}\} \langle C \rangle \{Q * \boxed{Q'}\}}$$

$$\frac{\begin{array}{c} p, q \text{ stable under } R \\ \emptyset, G \vdash \{p\} \langle C \rangle \{q\} \end{array}}{R, G \vdash \{p\} \langle C \rangle \{q\}}$$

grace logic

- programming language

- assertion language

- Hoare triples

- proof rules

grace logic

- programming language

- assertion language

- Hoare triples

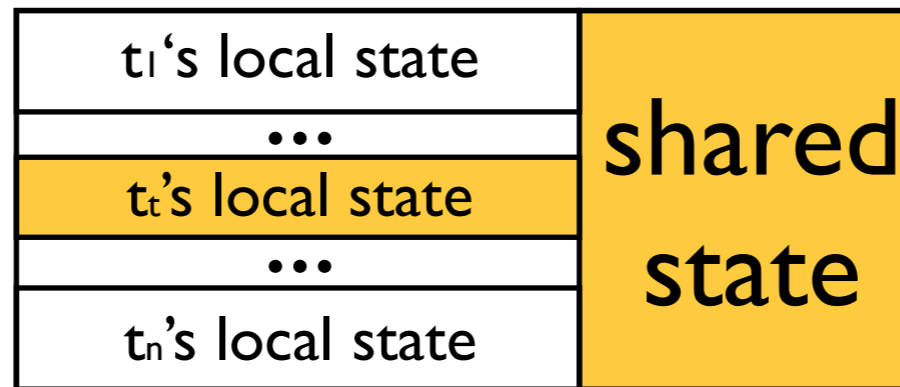
- proof rules

- RGSep + temporal assertions

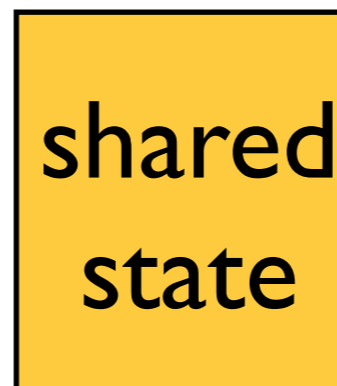
global view

SL

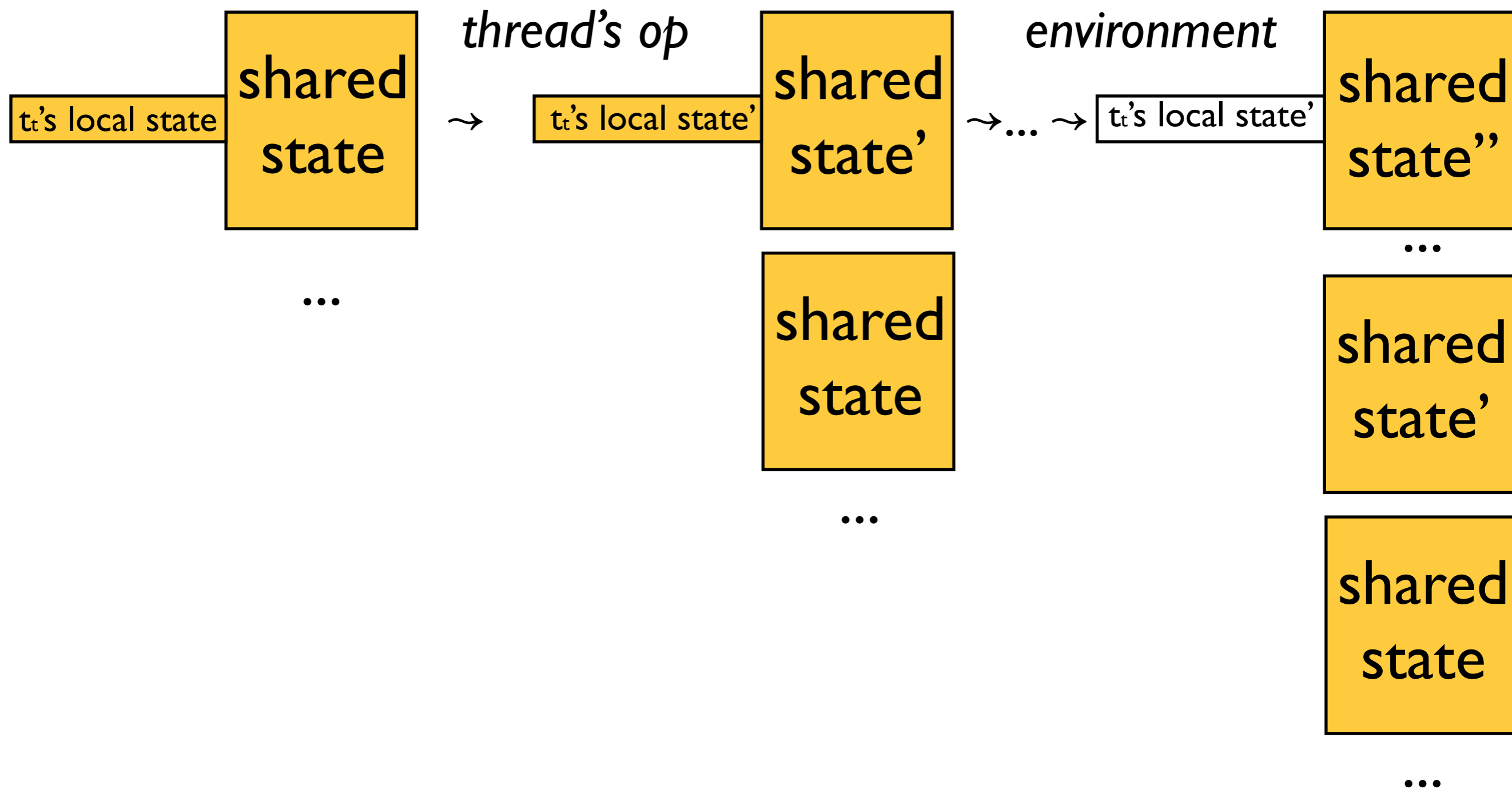
R/G



...



local view

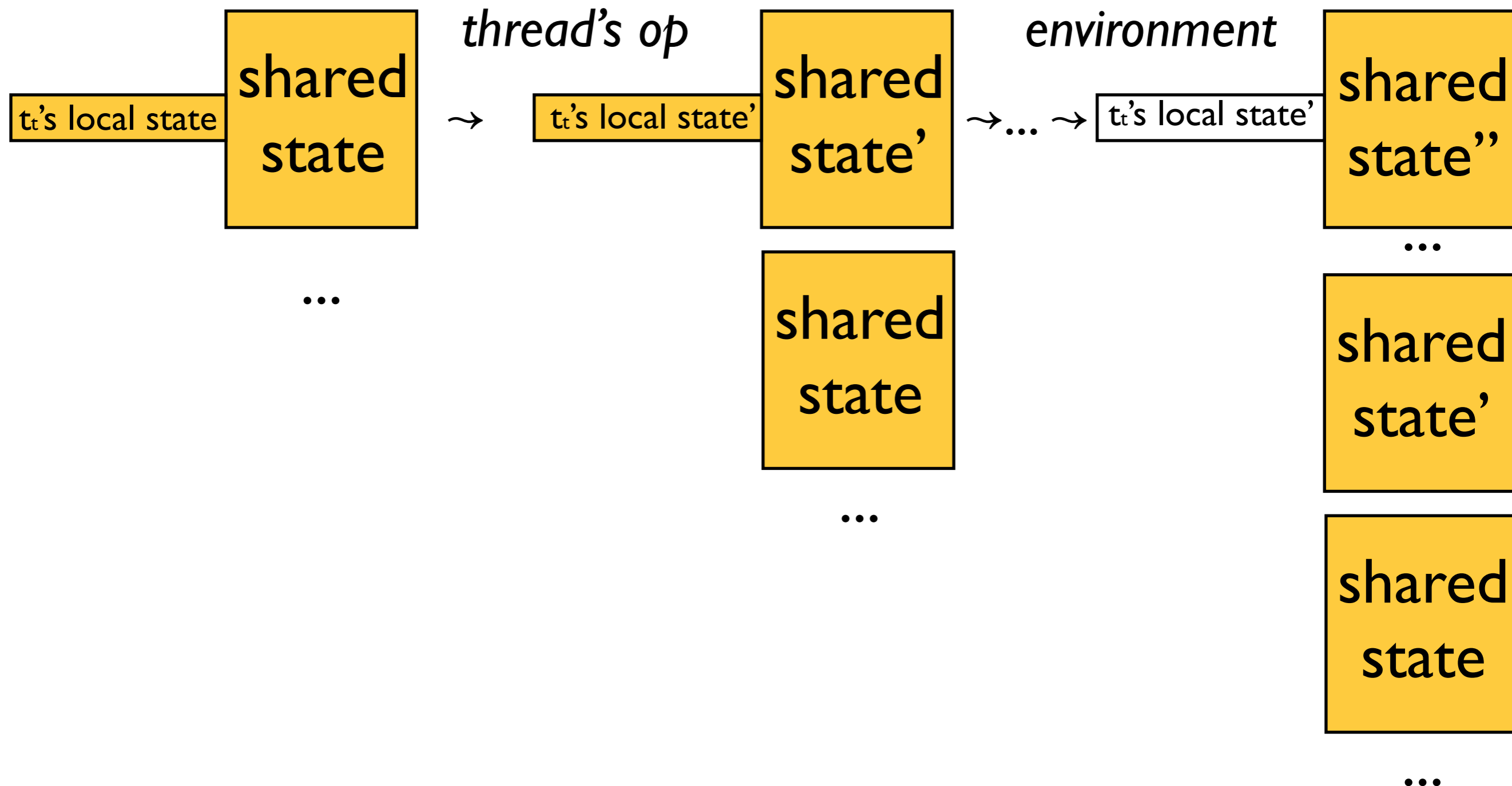


local view

$R_t, G_t \subseteq (\text{shared state}) \times (\text{shared state})$

assert $\bullet \leadsto \bullet \in G_t$

assume $\bullet \leadsto \bullet \in R_t$



assertion language

$\eta \in \text{History} = \text{State}^+;$

$\omega \in \text{World} = \{(\theta, \eta, \mathbf{i}) \in \text{State} \times \text{History} \times \text{LInt} \mid (\theta * \eta_{|\eta|}) \downarrow\}$

$\tau, \Upsilon ::= \boxed{p} \mid \text{true} \mid \neg \tau \mid \tau_1 \Rightarrow \tau_2 \mid \exists x. \tau \mid \tau_1 \text{ since } \tau_2 \mid \tau \triangleleft \boxed{p}$

$P, Q ::= p \mid \tau \mid \text{true} \mid \neg P \mid P \Rightarrow Q \mid \exists x. P \mid P * Q$

assertion language

$$\eta \in \text{History} = \text{State}^+;$$

$$\omega \in \text{World} = \{(\theta, \eta, \mathbf{i}) \in \text{State} \times \text{History} \times \text{LInt} \mid (\theta * \eta_{|\eta|}) \downarrow\}$$

$$\tau, \Upsilon ::= \boxed{p} \mid \text{true} \mid \neg \tau \mid \tau_1 \Rightarrow \tau_2 \mid \exists x. \tau \mid \tau_1 \text{ since } \tau_2 \mid \tau \triangleleft \boxed{p}$$

$$P, Q ::= p \mid \tau \mid \text{true} \mid \neg P \mid P \Rightarrow Q \mid \exists x. P \mid P * Q$$

$$\eta, \mathbf{i} \models \boxed{p} \iff \eta_{|\eta|}, \mathbf{i} \models p$$

$$\eta, \mathbf{i} \models \tau_1 \text{ since } \tau_2 \iff \exists i \in \{1, \dots, |\eta|\}. (\eta|_i, \mathbf{i} \models \tau_1) \wedge \\ \forall j \in \{i, \dots, |\eta|\}. (\eta|_j, \mathbf{i} \models \tau_2)$$

$$\eta, \mathbf{i} \models \tau \triangleleft \boxed{p} \iff \exists \eta'. \eta = \eta' \theta \wedge \eta', \mathbf{i} \models \tau \wedge \theta \models p$$

assertion language

$$\eta \in \text{History} = \text{State}^+;$$

$$\omega \in \text{World} = \{(\theta, \eta, \mathbf{i}) \in \text{State} \times \text{History} \times \text{LInt} \mid (\theta * \eta_{|\eta|}) \downarrow\}$$

$$\tau, \Upsilon ::= \boxed{p} \mid \text{true} \mid \neg \tau \mid \tau_1 \Rightarrow \tau_2 \mid \exists x. \tau \mid \tau_1 \text{ since } \tau_2 \mid \tau \triangleleft \boxed{p}$$

$$P, Q ::= p \mid \tau \mid \text{true} \mid \neg P \mid P \Rightarrow Q \mid \exists x. P \mid P * Q$$

$$\begin{aligned} \eta, \mathbf{i} \models \boxed{p} &\iff \eta_{|\eta|}, \mathbf{i} \models p \\ \eta, \mathbf{i} \models \tau_1 \text{ since } \tau_2 &\iff \exists i \in \{1, \dots, |\eta|\}. (\eta|_i, \mathbf{i} \models \tau_1) \wedge \\ &\quad \forall j \in \{i, \dots, |\eta|\}. (\eta|_j, \mathbf{i} \models \tau_2) \\ \eta, \mathbf{i} \models \tau \triangleleft \boxed{p} &\iff \exists \eta'. \eta = \eta' \theta \wedge \eta', \mathbf{i} \models \tau \wedge \theta \models p \end{aligned}$$

$$\begin{aligned} \theta, \eta, \mathbf{i} \models p &\iff \theta, \mathbf{i} \models p \\ \theta, \eta, \mathbf{i} \models \tau &\iff \eta, \mathbf{i} \models \tau \\ \theta, \eta, \mathbf{i} \models \exists x. P &\iff \exists u. (\theta, \eta, \mathbf{i}[x : u]) \models P \\ \theta, \eta, \mathbf{i} \models \forall x. P &\iff \forall u. (\theta, \eta, \mathbf{i}[x : u]) \models P \\ \theta, \eta, \mathbf{i} \models P * Q &\iff \exists \theta', \theta''. (\theta = \theta' * \theta'') \wedge \\ &\quad (\theta', \eta, \mathbf{i} \models P) \wedge (\theta'', \eta, \mathbf{i} \models Q) \end{aligned}$$

relies & guarantees

$$R_t, G_t \in \text{Action} = 2^{\text{State} \times \text{State} \times \text{State}}$$

$$l \mid p * X \rightsquigarrow q * X$$



p, q precise

stability

An assertion P is *stable* under an action $l \mid p \rightsquigarrow q$ and a temporal invariant Υ if it is insensitive to changes to the shared state permitted by the action that preserve the invariant:

$$\begin{aligned} & \forall \theta_l, \eta, \theta_s, \theta'_l, \theta'_s, \mathbf{i}. (\theta_l, \eta\theta_s, \mathbf{i} \models P) \wedge \\ & (\theta'_l, \mathbf{i}) \in \llbracket l \rrbracket \wedge (\theta_s, \mathbf{i}) \in \llbracket p \rrbracket \wedge (\theta'_s, \mathbf{i}) \in \llbracket q \rrbracket \wedge \\ & (\eta\theta_s, \mathbf{i}) \in \llbracket \Upsilon \rrbracket \wedge (\eta\theta_s\theta'_s, \mathbf{i}) \in \llbracket \Upsilon \rrbracket \wedge \\ & (\theta_l * \theta'_l * \theta_s) \downarrow \wedge (\theta_l * \theta'_s) \downarrow \implies (\theta_l, \eta\theta_s\theta'_s, \mathbf{i} \models P), \end{aligned}$$

stability

if A is stable $\implies A$ since B is stable

proof rules

$$\frac{R, G, \Upsilon \vdash \{P_1\} C_1 \{P_2\} \quad R, G, \Upsilon \vdash \{P_2\} C_2 \{P_3\}}{R, G, \Upsilon \vdash \{P_1\} C_1; C_2 \{P_3\}} \text{SEQ}$$

$$\frac{R, G, \Upsilon \vdash \{P\} C_1 \{Q\} \quad R, G, \Upsilon \vdash \{P\} C_2 \{Q\}}{R, G, \Upsilon \vdash \{P\} C_1 + C_2 \{Q\}} \text{CHOICE}$$

$$\frac{R, G, \Upsilon \vdash \{P\} C \{P\}}{R, G, \Upsilon \vdash \{P\} C^* \{P\}} \text{LOOP}$$

$$\frac{R, G, \Upsilon \vdash \{P_1\} C \{Q_1\} \quad R, G, \Upsilon \vdash \{P_2\} C \{Q_2\}}{R, G, \Upsilon \vdash \{P_1 \vee P_2\} C \{Q_1 \vee Q_2\}} \text{DISJ}$$

invariant introduction

$$\frac{R, G, \Upsilon \vdash \{P \wedge \Upsilon\} C \{Q\}}{R, G, \Upsilon \vdash \{P\} C \{Q\}}$$

shared proof rule

$$\frac{Q \Rightarrow \Upsilon \quad P, Q \text{ are stable under } R \text{ and } \Upsilon \quad \emptyset, G, \text{true} \vdash_{\text{tid}} \{P\} \langle C \rangle_a \{Q\}}{R, G, \Upsilon \vdash_{\text{tid}} \{P\} \langle C \rangle_a \{Q\}} \quad \text{SHARED-R}$$

$$\frac{p \Rightarrow l * \text{true} \quad \{l \mid p_s \rightsquigarrow q_s\} \Rightarrow \{a\} \quad a \in G \quad \emptyset, \emptyset, \text{true} \vdash_{\text{tid}} \{p * p_s\} C \{q * q_s\}}{\emptyset, G, \text{true} \vdash_{\text{tid}} \{p \wedge \tau \wedge \boxed{p_s}\} \langle C \rangle_a \{q \wedge ((\tau \wedge \boxed{p_s}) \triangleleft \boxed{q_s})\}} \quad \text{SHARED}$$

specialized shared rule

$$\frac{\begin{array}{l} p \Rightarrow l * \text{true} \quad a = (l \mid p'_s \rightsquigarrow q'_s) \in G \quad p_s \Rightarrow p'_s \quad q_s \Rightarrow q'_s \\ \emptyset, \emptyset, \text{true} \vdash_{\text{tid}} \{p * (p_s \wedge \neg(g \wedge r))\} C \{q * (q_s \wedge (g \wedge r \Rightarrow c))\} \\ \emptyset, \emptyset, \text{true} \vdash_{\text{tid}} \{p * (p_s \wedge g \wedge c)\} C \{q * (q_s \wedge (g \Rightarrow c))\} \end{array}}{\begin{array}{l} \emptyset, G, \text{true} \vdash_{\text{tid}} \{p \wedge \boxed{p_s} \wedge ((\boxed{g} \text{ since } \boxed{r}) \Rightarrow \boxed{c})\} \\ \langle C \rangle_a \{q \wedge \boxed{q_s} \wedge ((\boxed{g} \text{ since } \boxed{r}) \Rightarrow \boxed{c})\} \end{array}}$$

temporal tautologies

$$(A \wedge B) \implies (A \text{ since } B)$$

$$((A \text{ since } B) \triangleleft A) \implies (A \text{ since } B)$$

$$(A \text{ since } B) \implies A$$

$$(\neg A \wedge \neg B) \implies \neg(A \text{ since } B)$$

$$\neg(A \text{ since } B) \triangleleft (\neg A \vee \neg B) \implies \neg(A \text{ since } B)$$

proof (retire)

```
1 {V ⊨ p ↦m - * Ftid ∧ [H * ∃y. C ↦ y * y ↦ - * true] ∧ [p ↦e - * true]}
2 void retire(int* p) {
3   {V ⊨ p ↦m - * ∃A. detached ↦ A * D(A) ∧ [H * ∃y. C ↦ y * y ↦ - * true] ∧ [p ↦e - * true]}
4   insert(detached, p);
5   {V ⊨ ∃A. detached ↦ A * D(A) ∧ [H * ∃y. C ↦ y * y ↦ - * true]}
6   if (nondet())
7     {V ⊨ Ftid ∧ [H * ∃y. C ↦ y * y ↦ - * true]}
8   return;
9   Set used;
10  {V ⊨ ∃A. detached ↦ A * D(A) ∧ used = ∅ ∧ [H * ∃y. C ↦ y * y ↦ - * true]}
11  while (!isEmpty(detached)) {
12    {V ⊨ ∃A. detached ↦ A * D(A) * D(used) ∧ A ≠ ∅ ∧ [H * ∃y. C ↦ y * y ↦ - * true]}
13    bool my = true;
14    Node *n = pop(detached);
15    {V ⊨ my ∧ ∃A. detached ↦ A * D(A) * D(used) * n ↦m - * [n ↦e - * true] * [H * ∃y. y ≠ n ∧ C ↦ y * y ↦ - * true]}
16    for (int i = 0; i < N && my; i++) {
17      {V ⊨ my ∧ 0 ≤ i < N ∧ ∃A. detached ↦ A * D(A) * D(used) * n ↦m - * [n ↦e - * true] *
18        [H * ∃y. y ≠ n ∧ C ↦ y * y ↦ - * true] ∧ ∀0 ≤ j < i. [∃y. y ≠ n ∧ C ↦ y * y ↦ - * true] since [HP[i] ≠ n * true]}
19      if ((HP[i] == n)ld)
20        my = false;
21      {V ⊨ 0 ≤ i < N ∧ ∃A. detached ↦ A * D(A) * D(used) * n ↦m - * [n ↦e - * true] *
22        [H * ∃y. y ≠ n ∧ C ↦ y * y ↦ - * true] ∧ (my ⇒ ∀0 ≤ j ≤ i. [∃y. y ≠ n ∧ C ↦ y * y ↦ - * true] since [HP[j] ≠ n * true])}
23    }
24    {V ⊨ ∃A. detached ↦ A * D(A) * D(used) * n ↦m - * [n ↦e - * true] ∧
25      [H * ∃y. C ↦ y * y ↦ - * true] ∧ (my ⇒ ∀0 ≤ j ≤ N. [∃y. y ≠ n ∧ C ↦ y * y ↦ - * true] since [HP[j] ≠ n * true])}
26    if (my) {
27      ⟨ ; ⟩Take;
28      {V ⊨ ∃A. detached ↦ A * D(A) * D(used) * n ↦m - * [H * ∃y. C ↦ y * y ↦ - * true]}
29      free(n);
30    } else {
31      insert(used, n);
32    }
33    {V ⊨ ∃A. detached ↦ A * D(A) * D(used) ∧ [H * ∃y. C ↦ y * y ↦ - * true]}
34  }
35  {V ⊨ detached ↦ ∅ * D(used) ∧ [H * ∃y. C ↦ y * y ↦ - * true]}
36  moveAll(detached, used);
37  {V ⊨ Ftid ∧ [H * ∃y. C ↦ y * y ↦ - * true]}
38 }
```

related work

- Hoare logic [Hoare, CACM'69]
- Rely/Guarantee reasoning [Jones, IFIP'83]
- Separation Logic [Reynolds, LICS'02]
- Concurrent separation logic [O'Hearn, TCS'07]
- RGSep [Vafeiadis, PhD'08]
- Temporal Separation Logic [Fu et al., CONCUR'10]
- Grace Logic [Gotsman, R, Yang, submitted]

summary

- idioms can simplify reasoning
 - grace period
 - hindsight [PODC'10]
- temporal reasoning can be natural for fine-grained concurrent algorithms
- program logic is a convenient formalization tool

the end

(c) some slides are by Viktor Vafeiadis

safety of manual concurrent memory reclamation algs.

- **RCU** [McKenney Slingwine'98]
- **Hazard Pointers** [Michael'02]
- **Pass the Buck** [Herlihy Luchangco Moir'02]
- **Epoch** [Fraser Haris'03]

$A \text{ since } (B \wedge A) \Rightarrow C$