Can Logic Tame Systems Programs?

Cristiano Calcagno
Imperial College
Wroclaw, July 2007

Joint work with Josh Berdine, Dino Distefano, Peter O'Hearn, Matthew Parkinson, Viktor Vafeiadis, Hongseok Yang

Beginning: Smallfoot

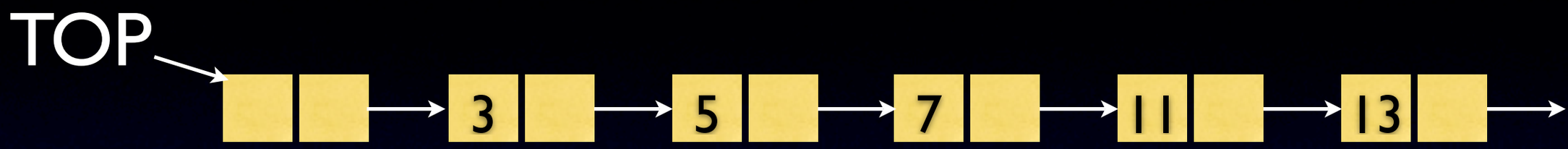
- Toy language
- Symbolic execution over SL fragment
- Annotations: pre/post and loop invariants
- Hard-coded predicates: list segments, trees (complete proof theory no induction)
- Concurrency: explicit || and ccr's

Systems programs?

- Handling C (current project, using CIL)
- Inferring annotations: shape analysis
- Composite data structures (CAV): predicate discovery, parameterised lists
- Open code/modularity: footprint analysis
- Interprocedural analysis
- (non-blocking) concurrency

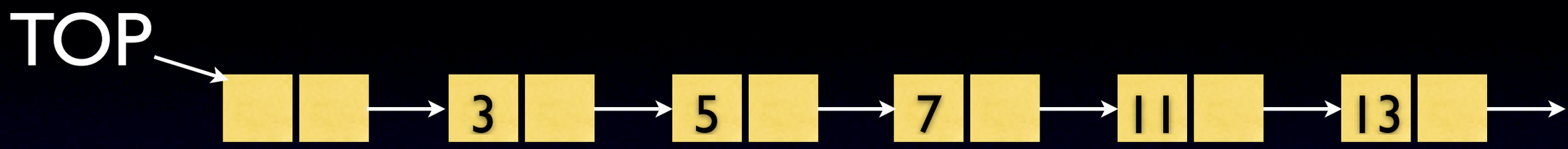
Non-Blocking Stack

Non-Blocking Stack




```
push(e) {
  local y,n,b;
  y = new();
  y->val = e;
  b=0;
  while(b==0) {
    n = TOP->tl;
    y->tl = n;
    b = CAS(TOP->tl,n,y); }
}
```

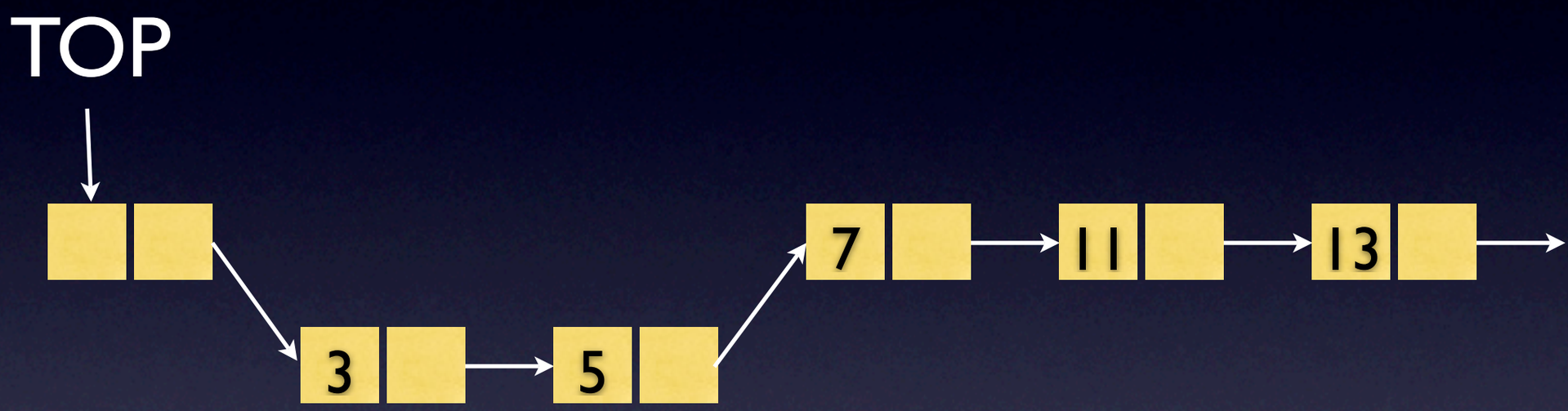
Non-Blocking Stack



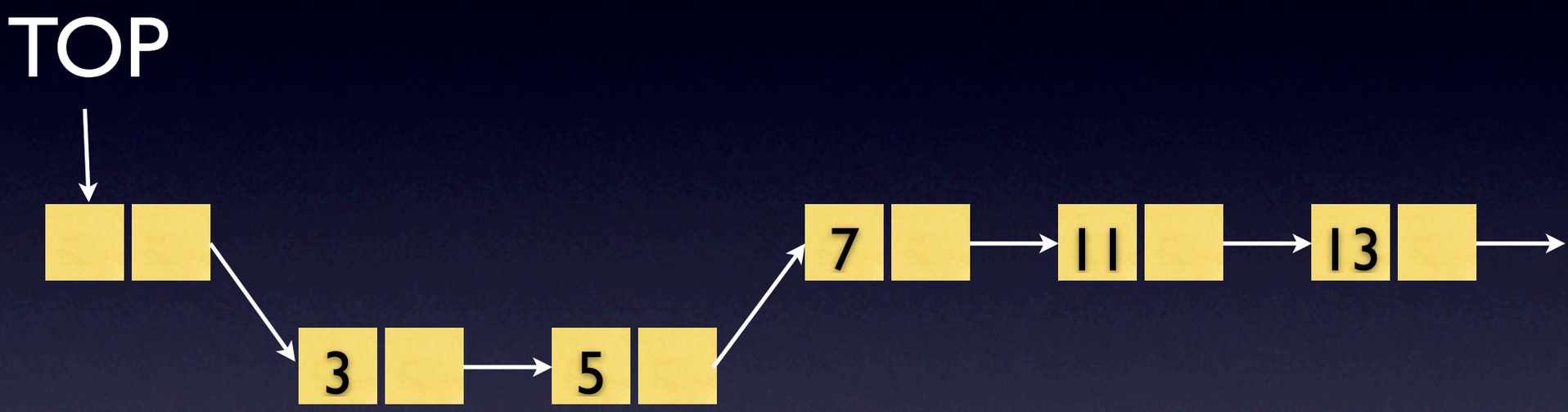

```
push(e) {
  local y,n,b;
  y = new();
  y->val = e;
  b=0;
  while(b==0) {
    n = TOP->tl;
    y->tl = n;
    b = CAS(TOP->tl,n,y); }
}
```

```
pop() {
  local y,z,b;
  b=0;
  while(b==0) {
   atomic when (TOP->tl != 0)
       { y = TOP->tl; z = y->tl; }
   b = CAS(TOP->tl,y,z); }
  ret = y->val;
  dispose y;
}
```

Non-Blocking Stack

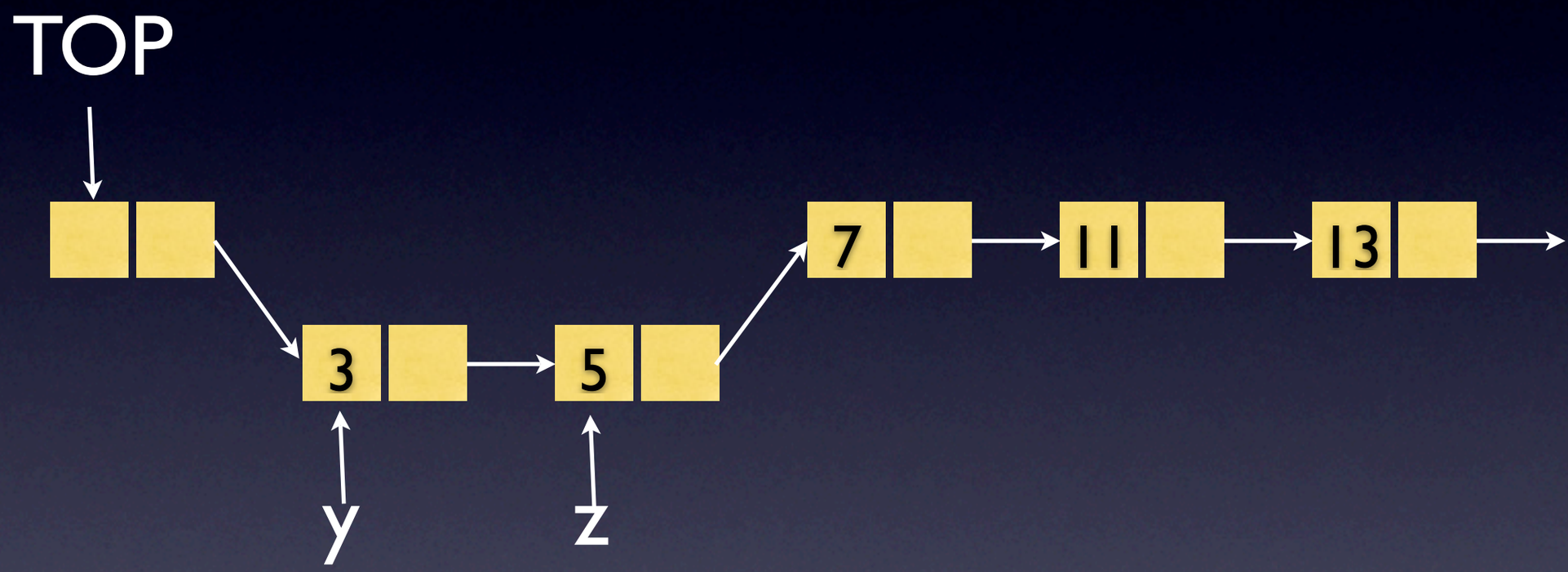


Non-Blocking Stack



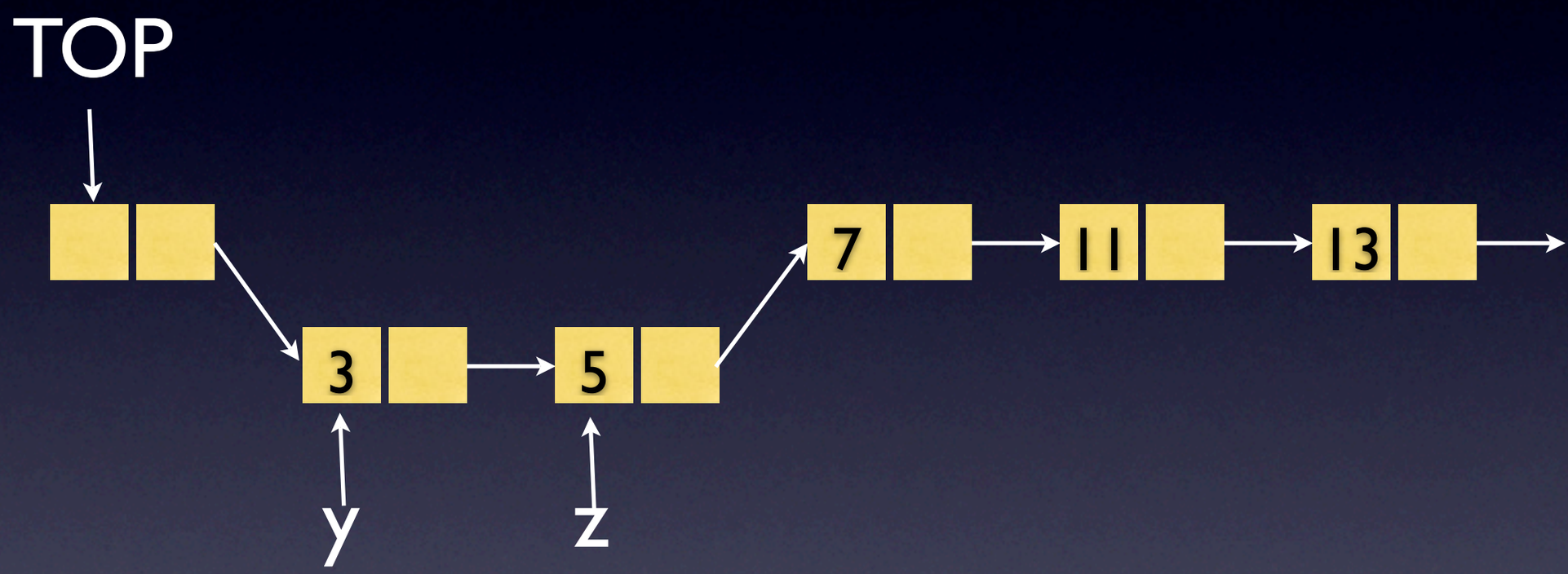
y = TOP->tl; z = y->tl;

Non-Blocking Stack

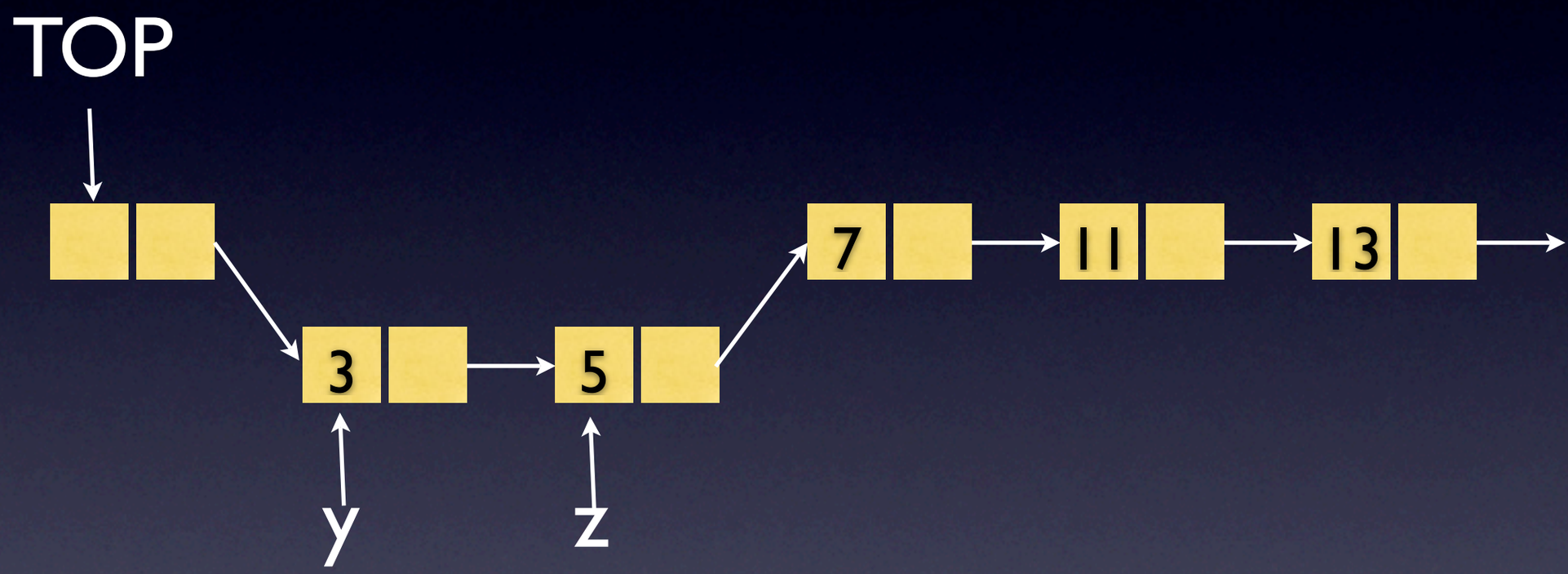


y = TOP->tl; z = y->tl;

Non-Blocking Stack

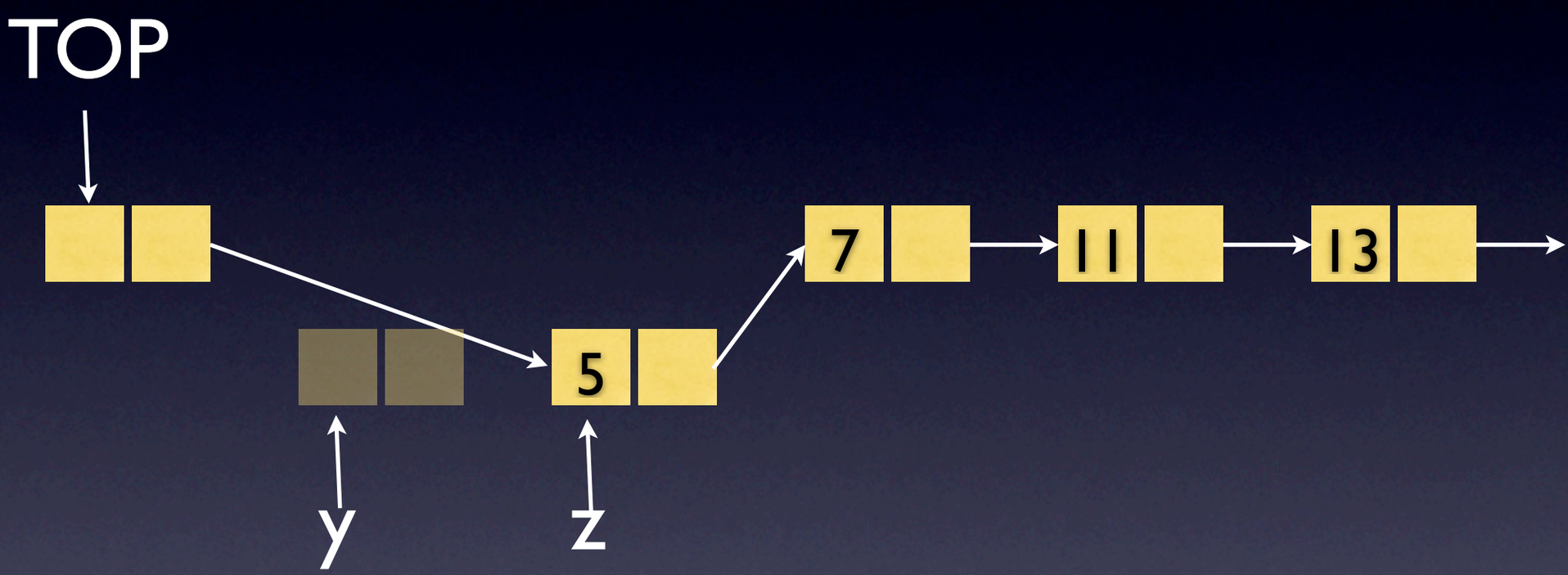


Non-Blocking Stack



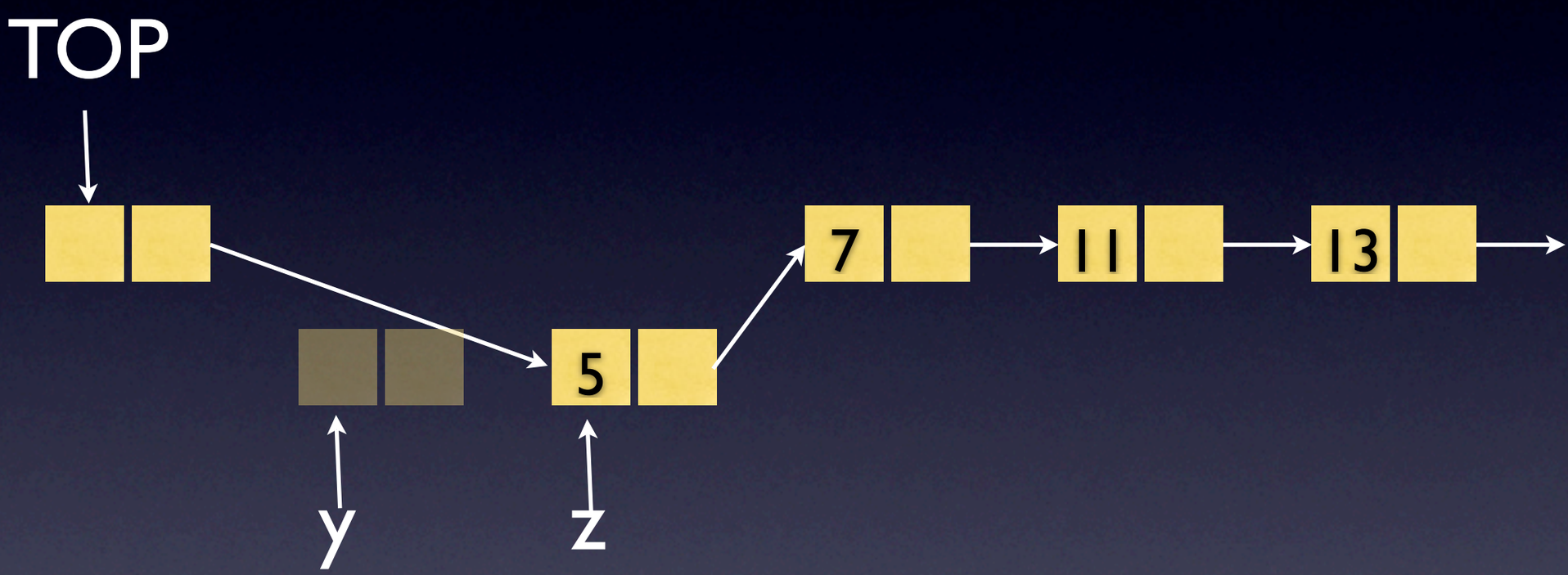
pop();

Non-Blocking Stack

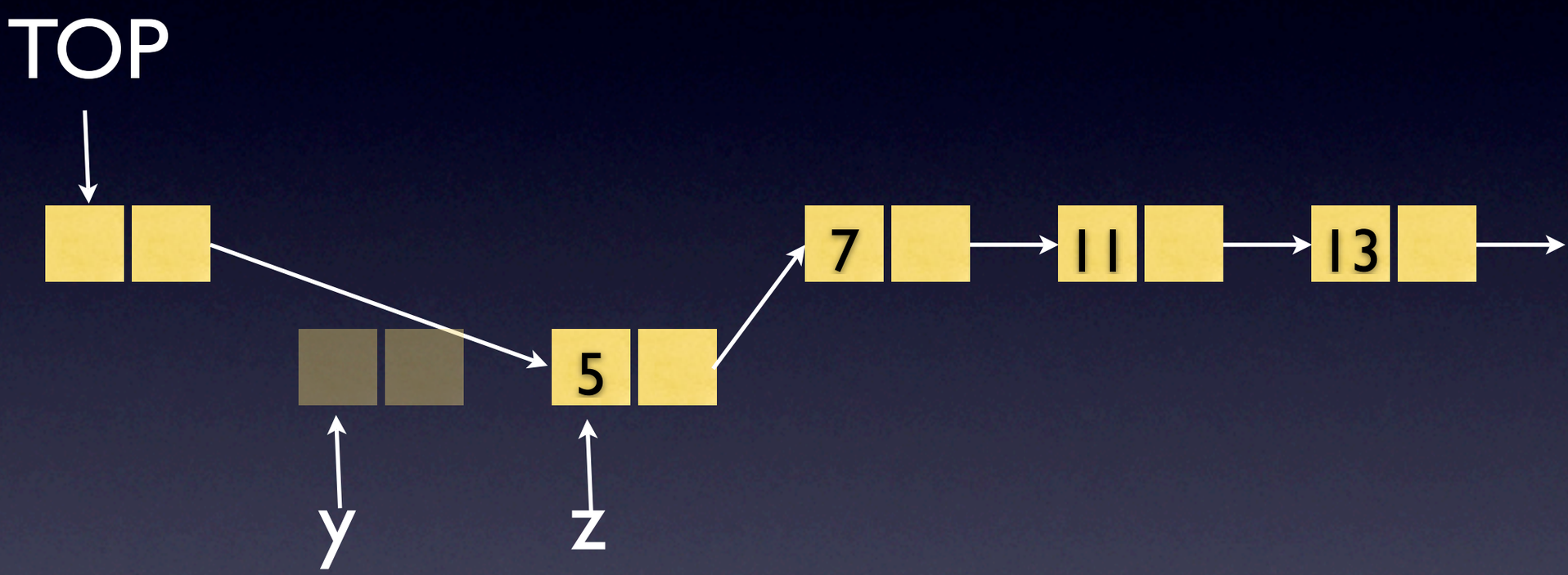


pop();

Non-Blocking Stack

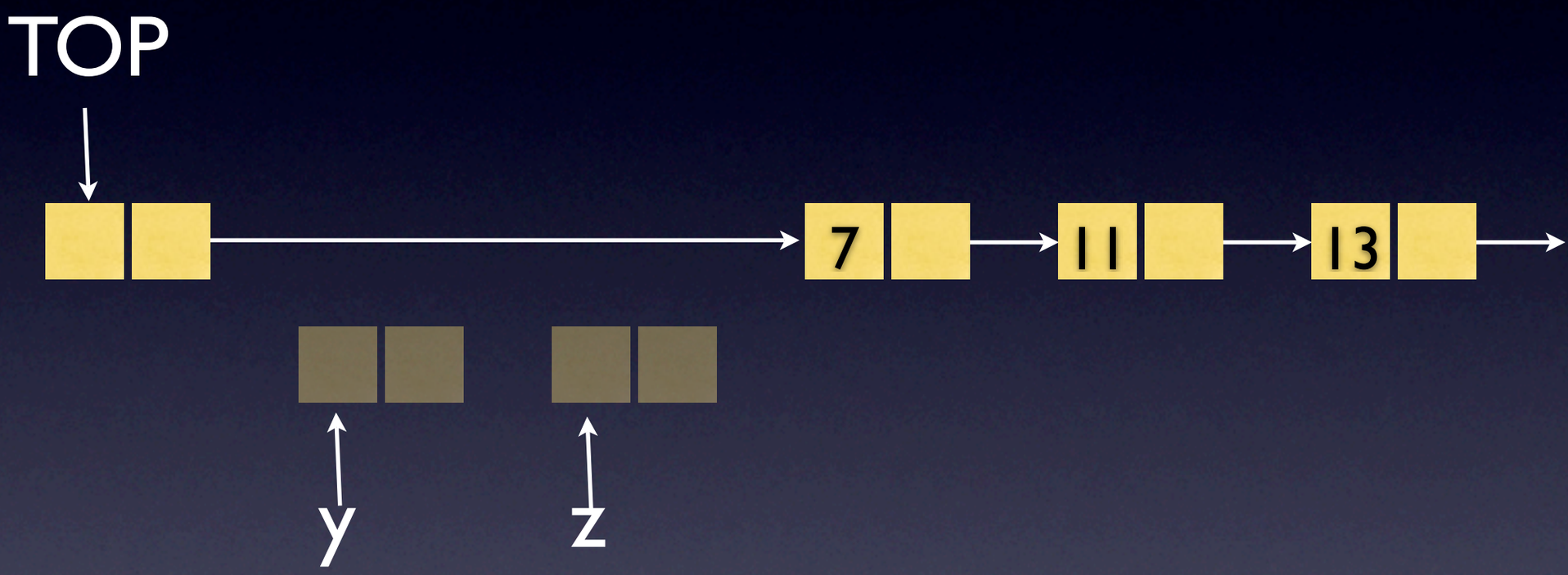


Non-Blocking Stack



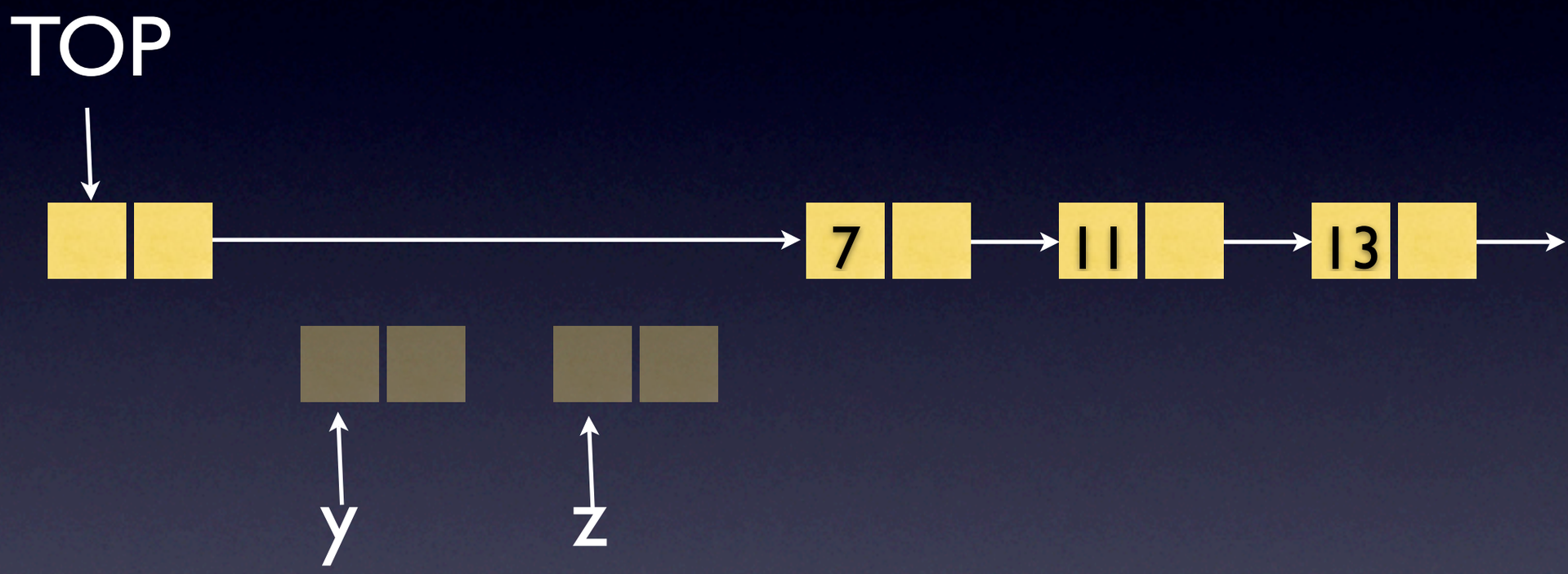
pop();

Non-Blocking Stack

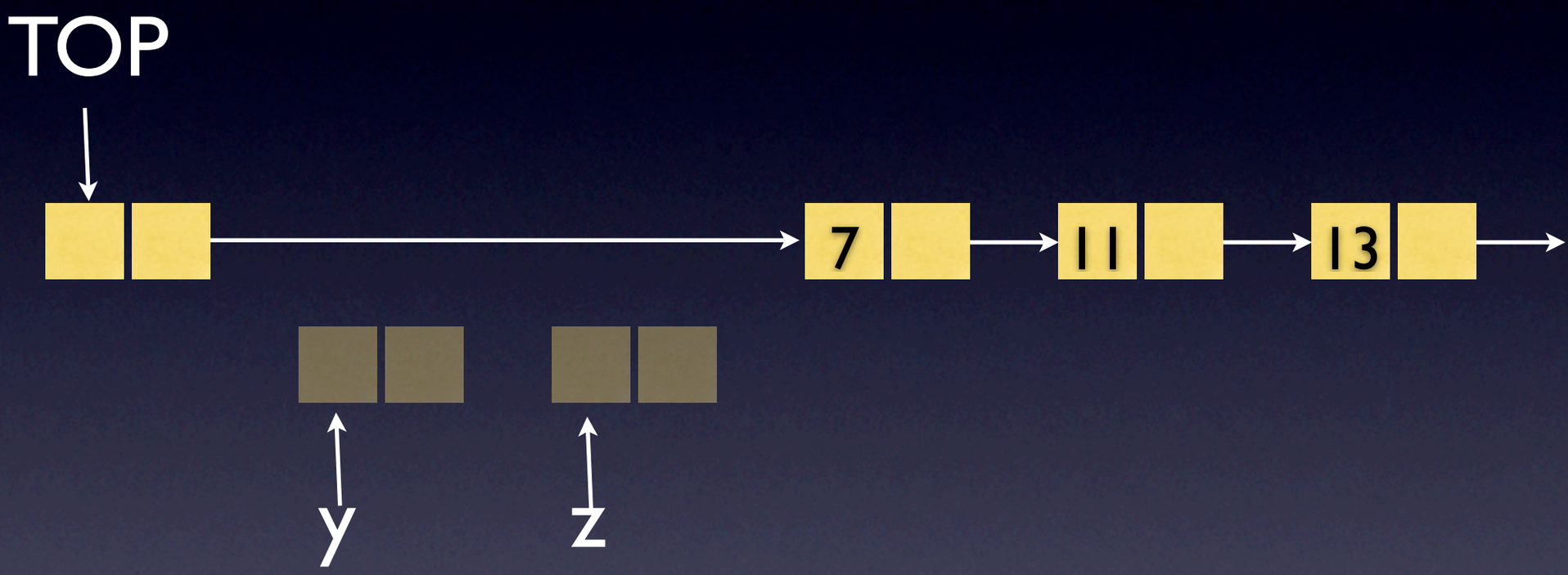


pop();

Non-Blocking Stack

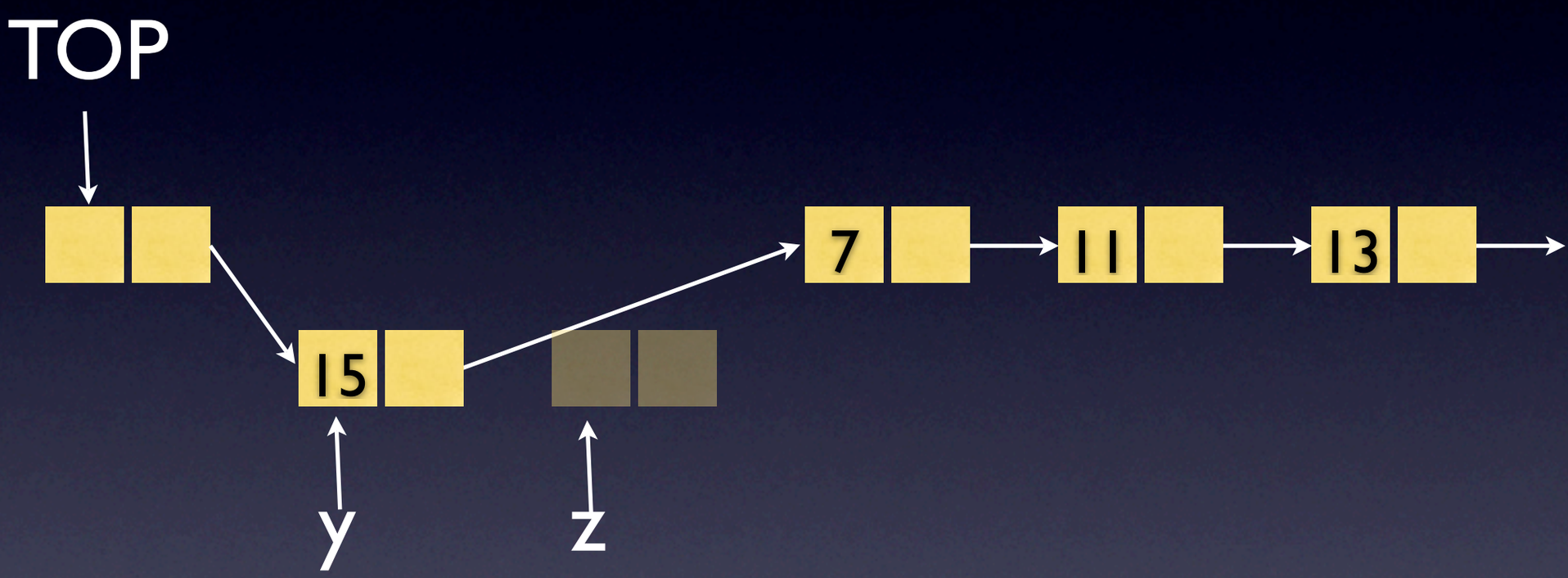


Non-Blocking Stack



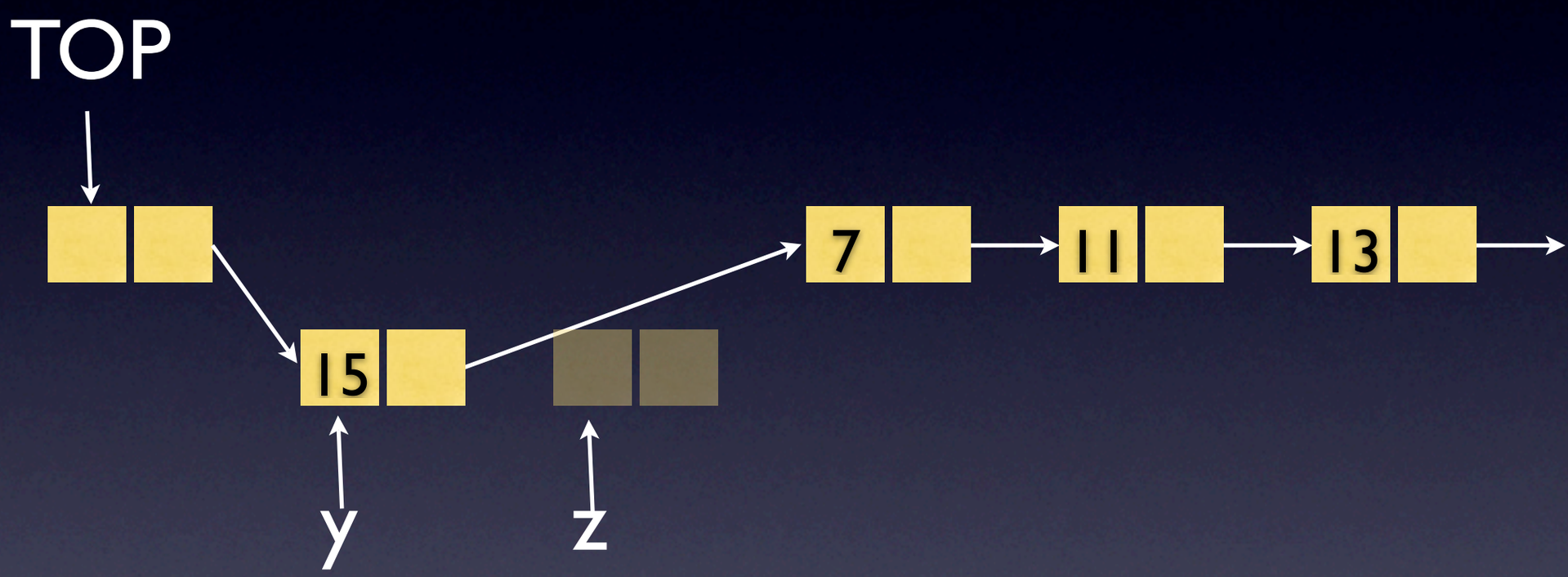
push(15);

Non-Blocking Stack

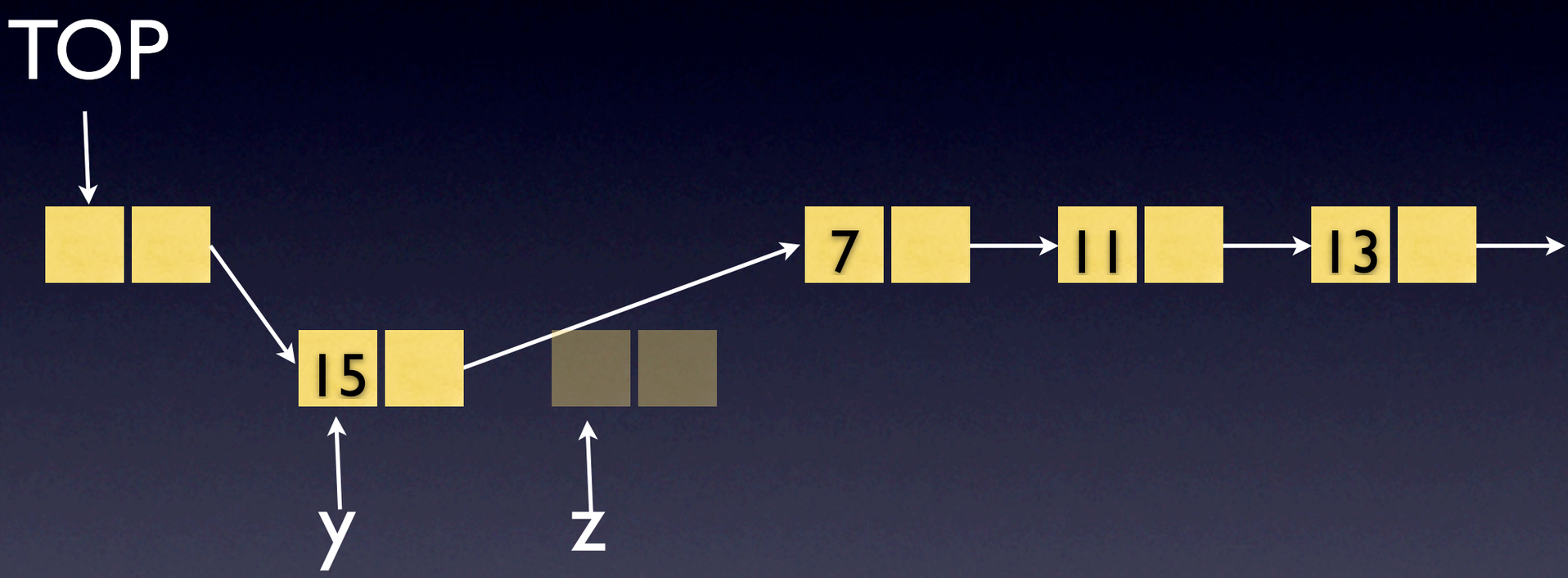


push(15);

Non-Blocking Stack

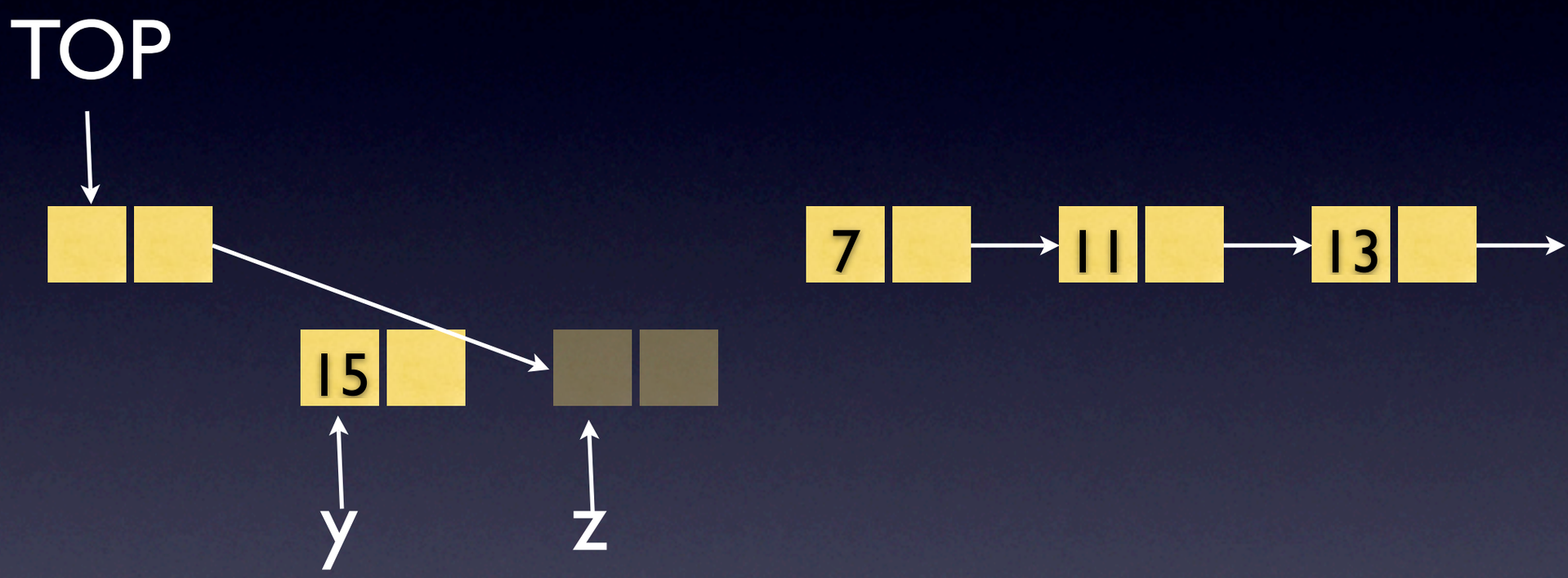


Non-Blocking Stack



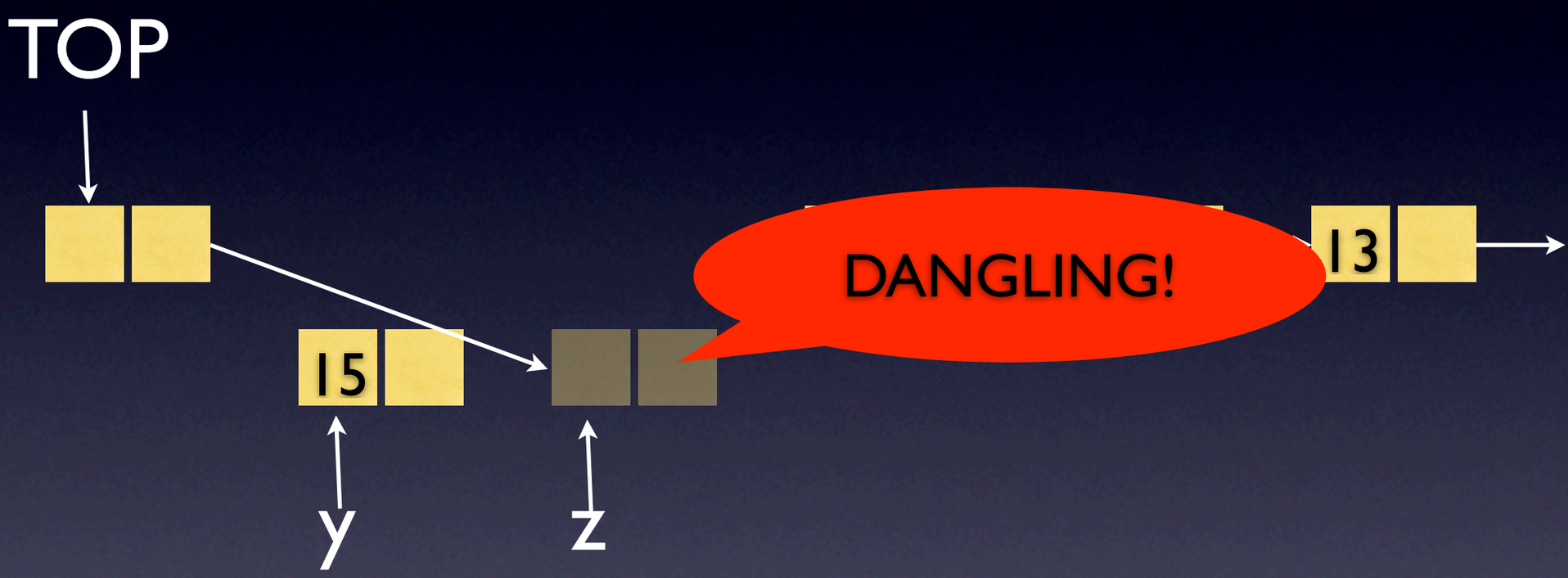
b = CAS(TOP->tl,y,z);

Non-Blocking Stack



b = CAS(TOP->tl,y,z);

Non-Blocking Stack



b = CAS(TOP->tl,y,z);

Reasoning

Separation logic: separation

Rely-Guarantee: interference

Parkinson and Vafeiadis: SL + RG

- local and shared state
- actions

Reasoning

Separation logic: separation

Rely-Guarantee: interference

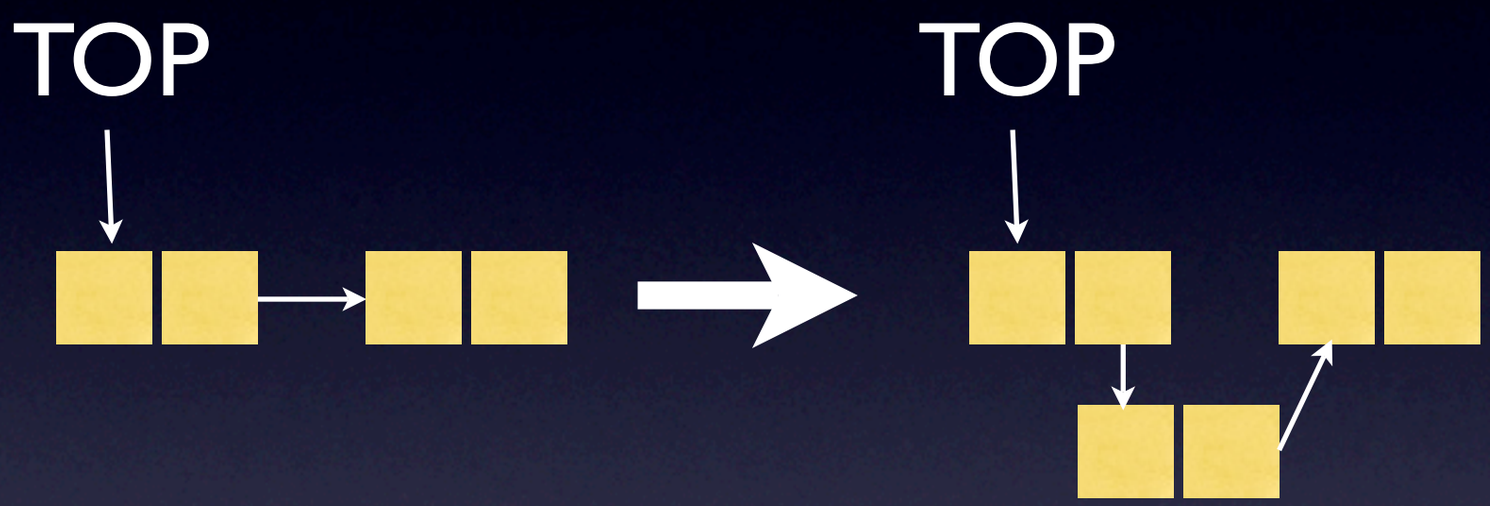
Parkinson and Vafeiadis: SL + RG

- local and shared state
- actions

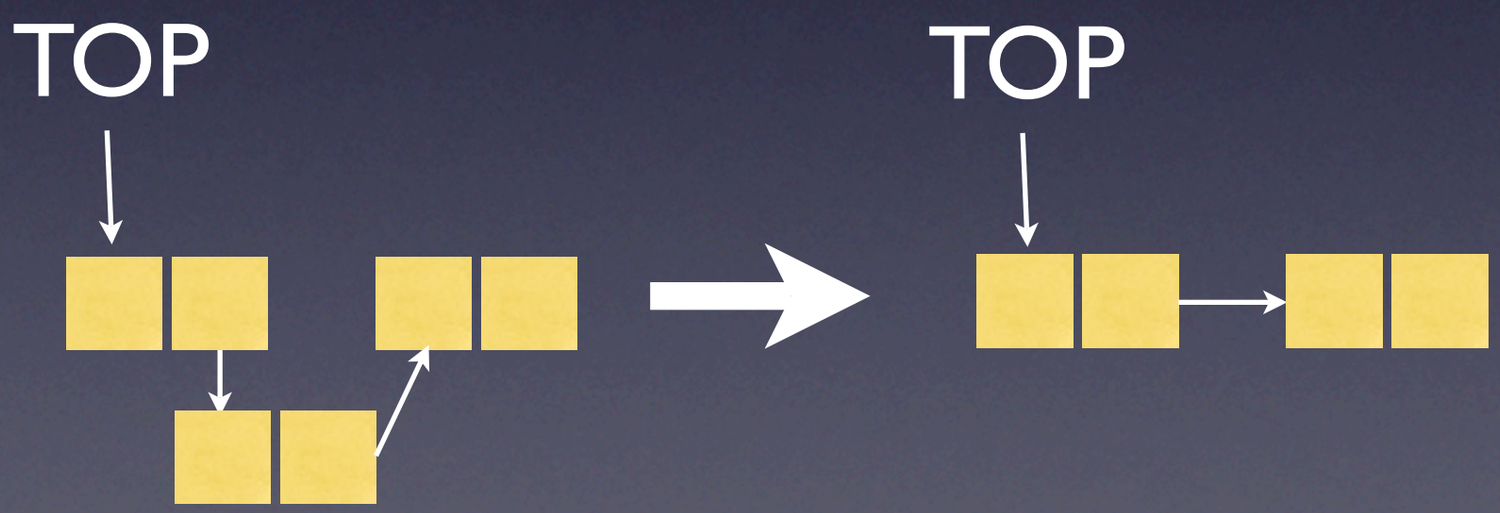
Tool: SmallfootRG

Operations

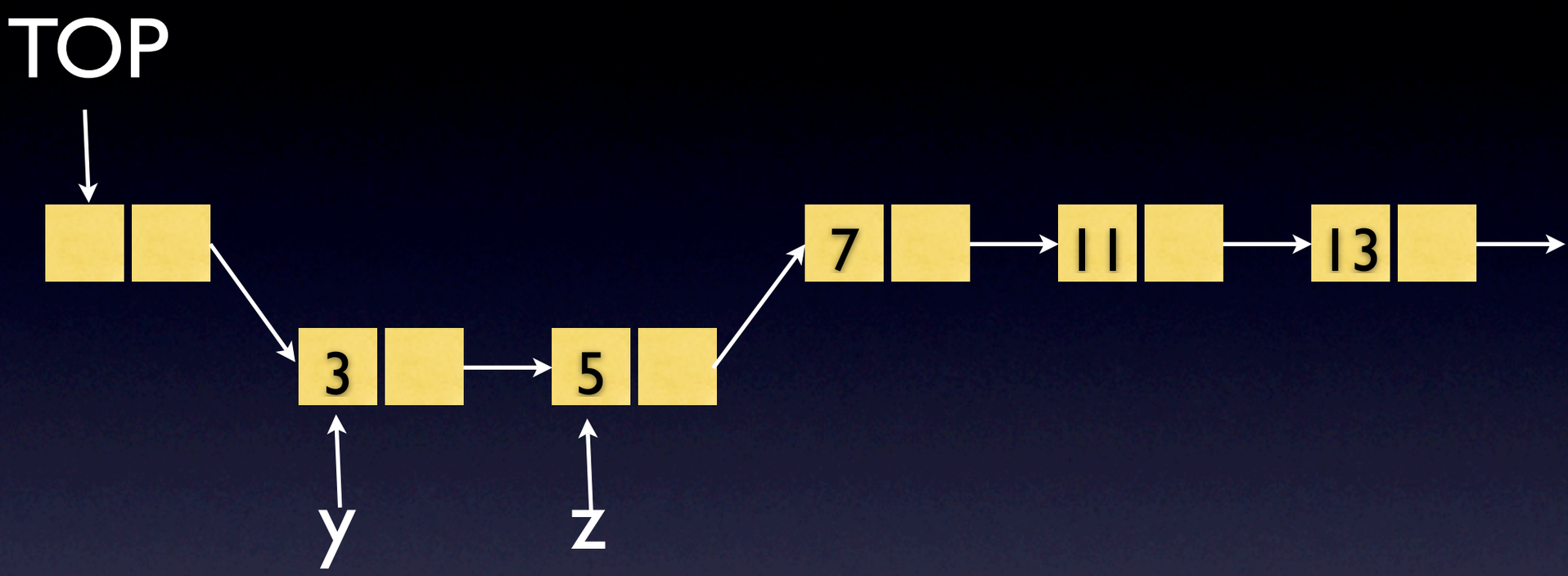
Push



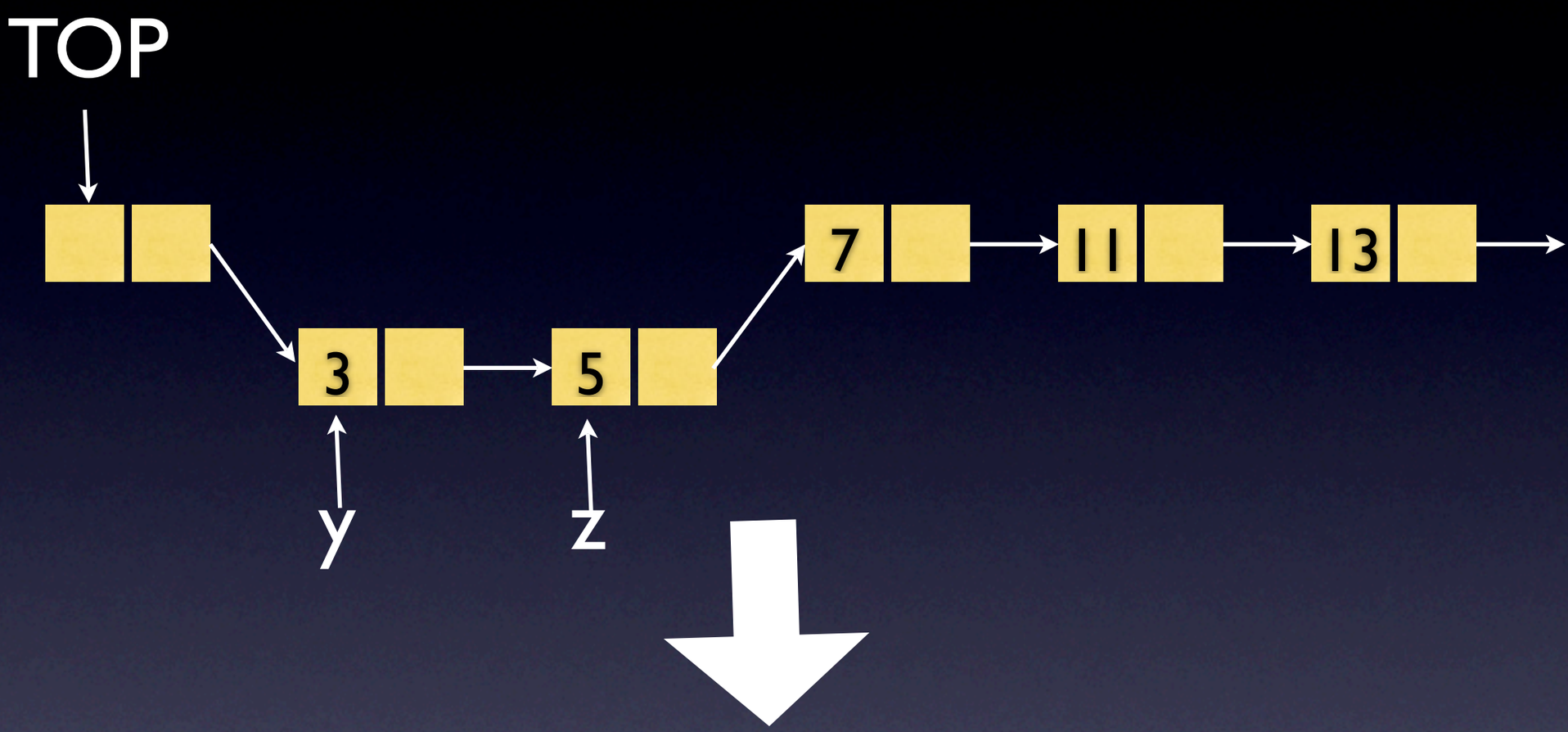
Pop



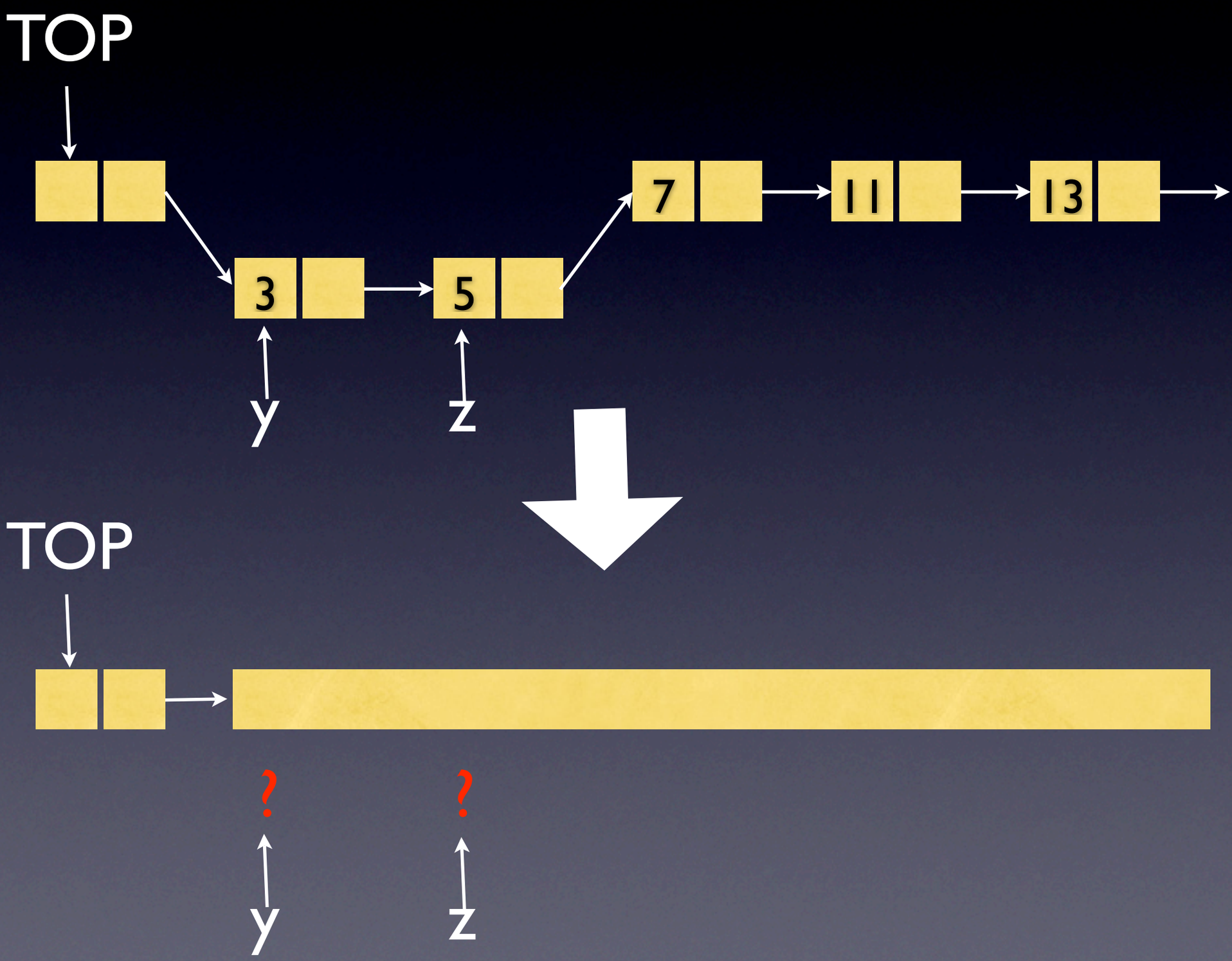
Interference



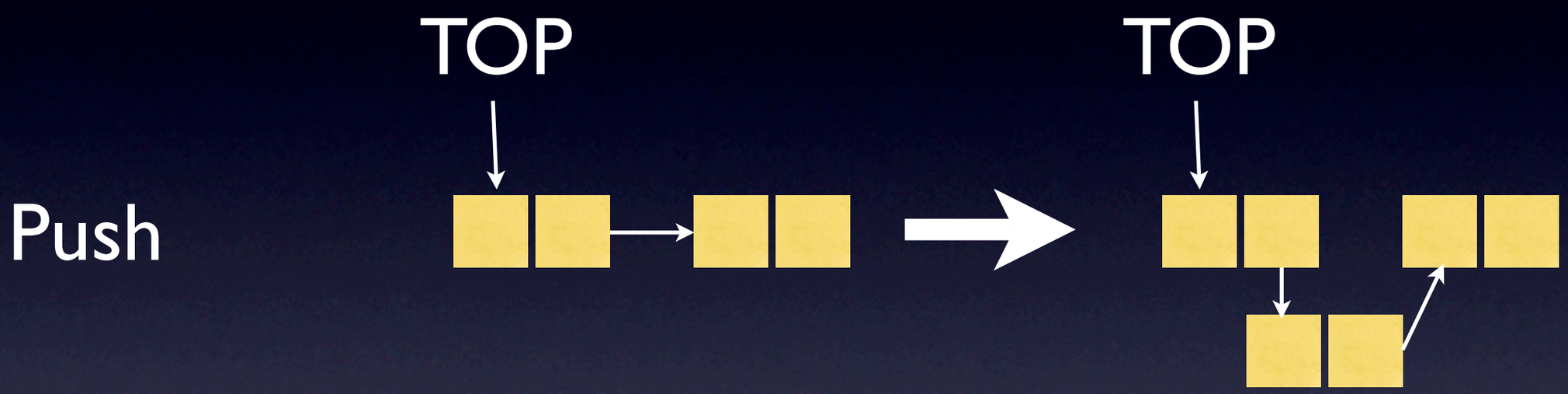
Interference



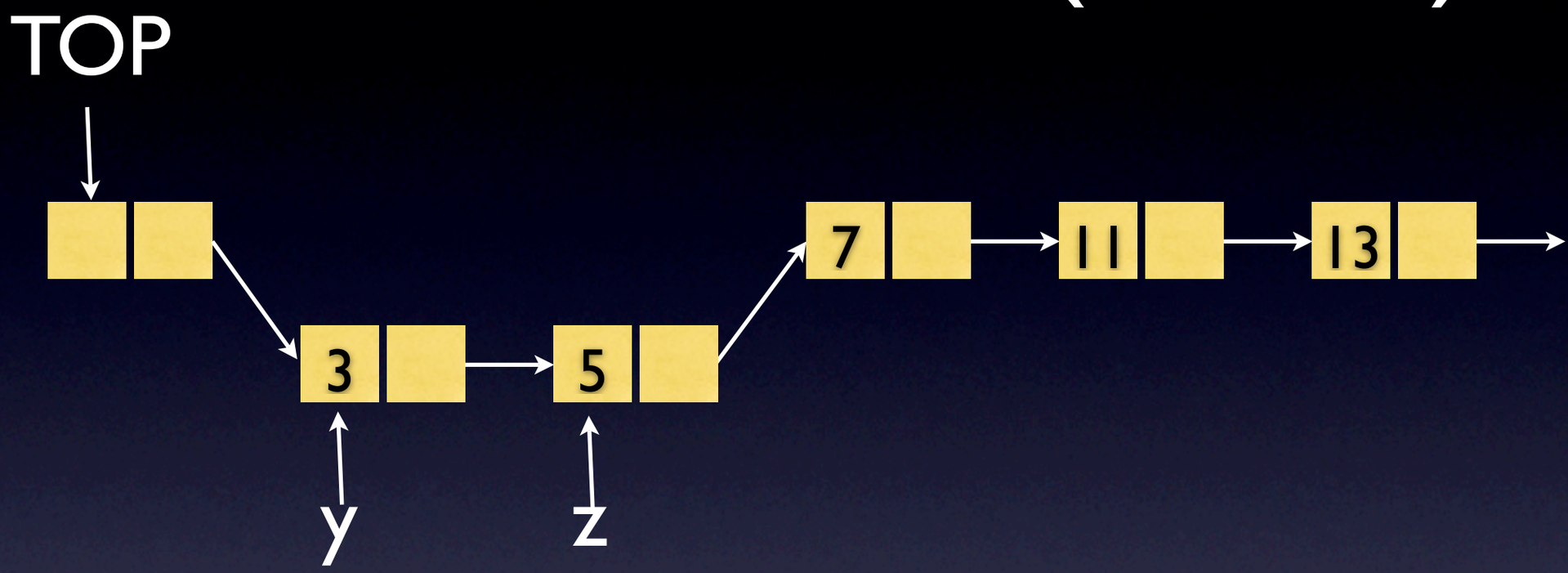
Interference



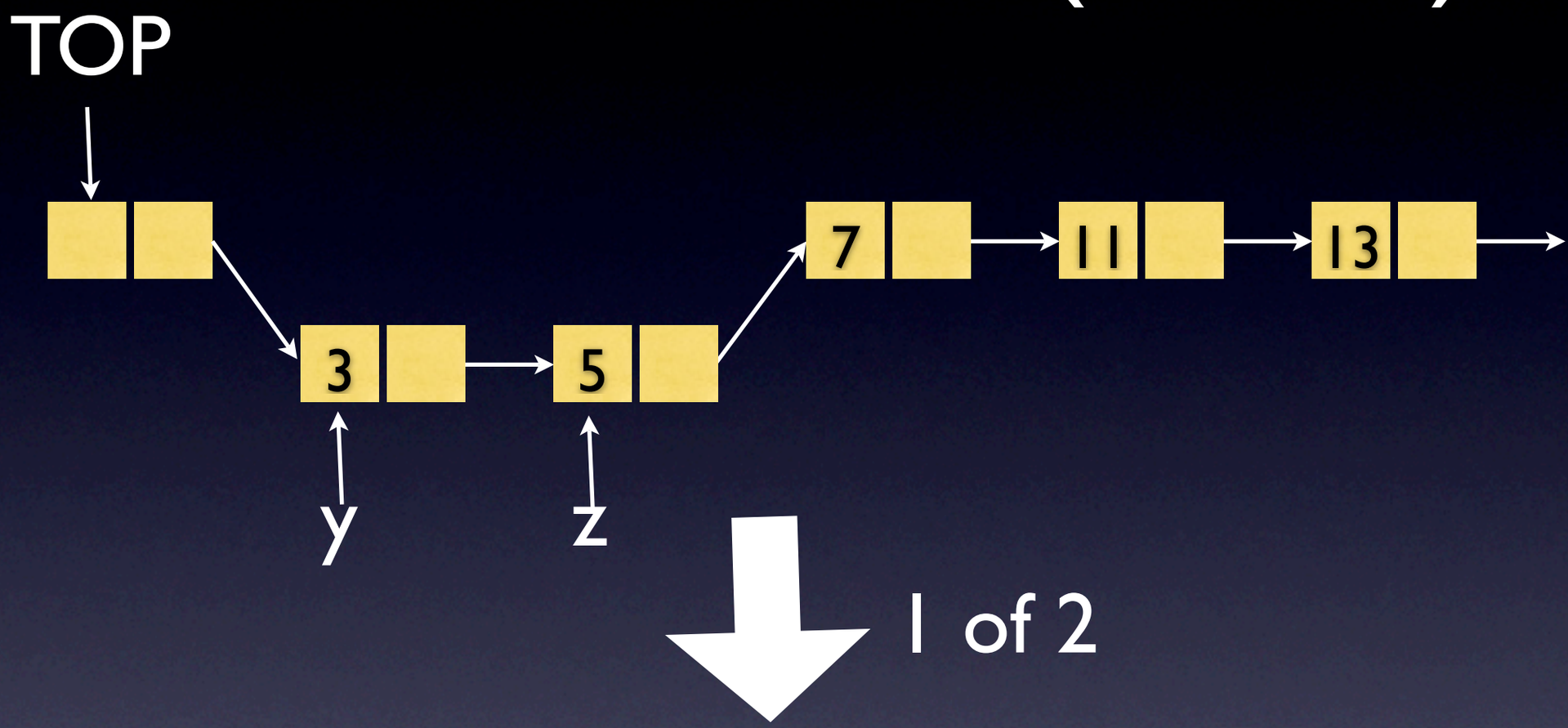
Operations (leaks)



Interference (leaks)

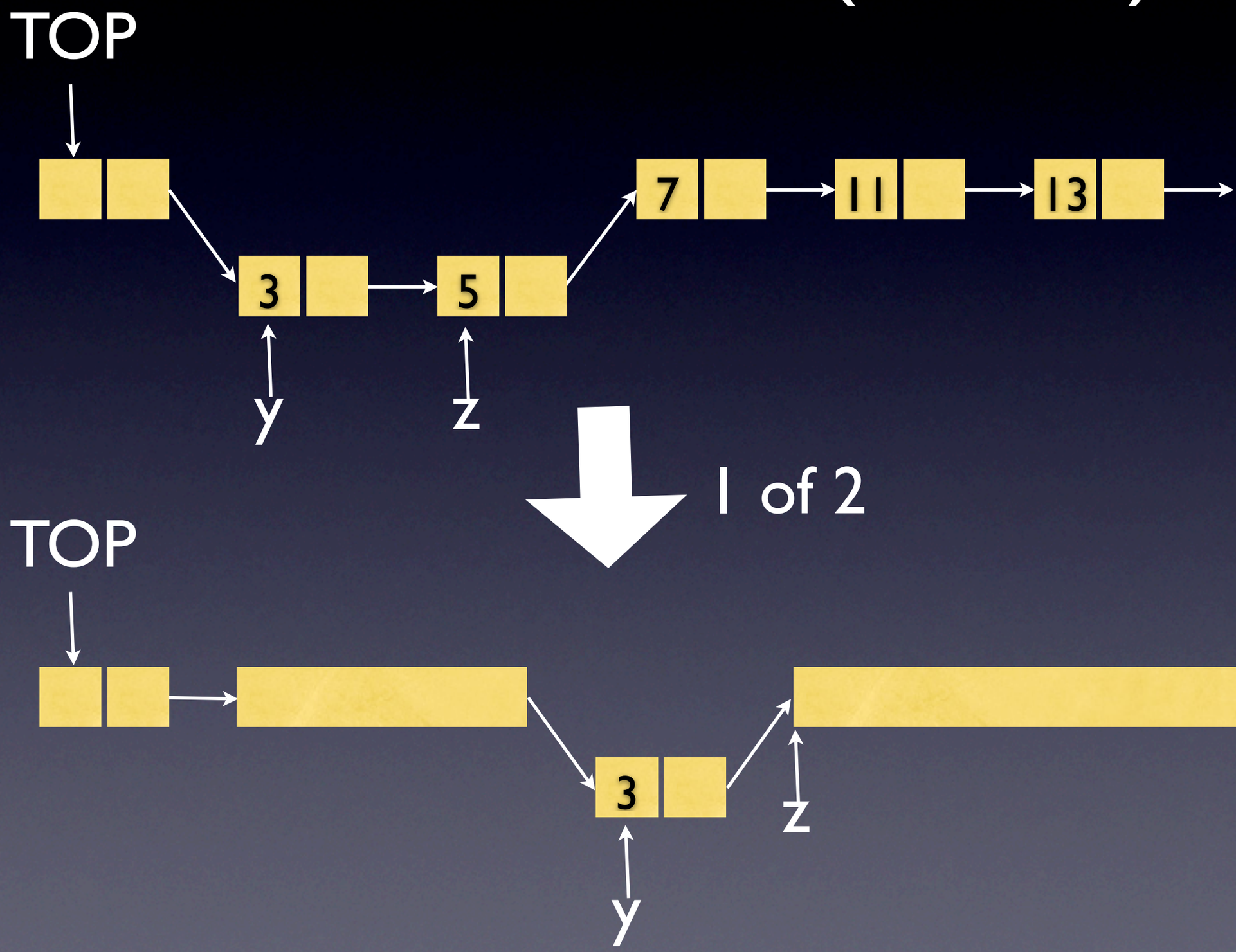


Interference (leaks)

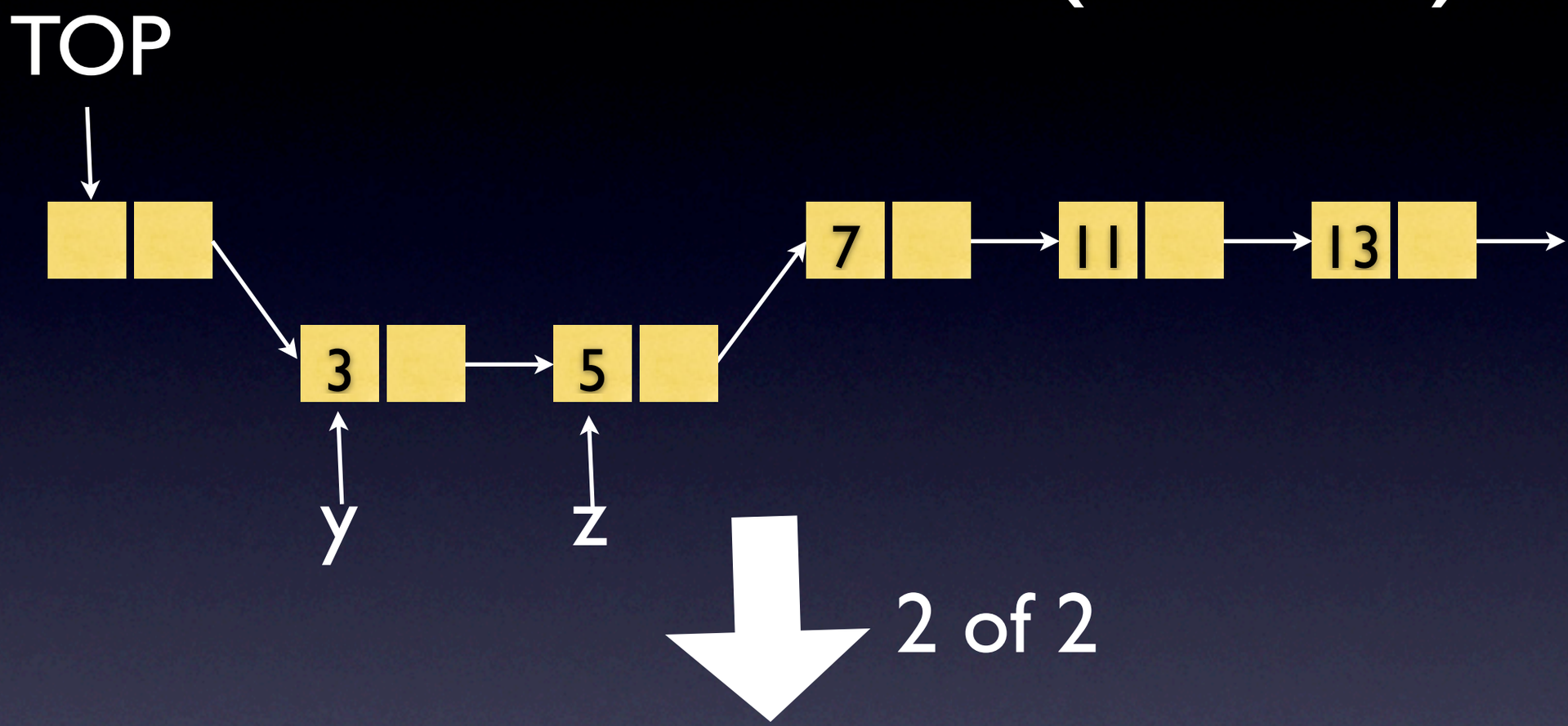


1 of 2

Interference (leaks)

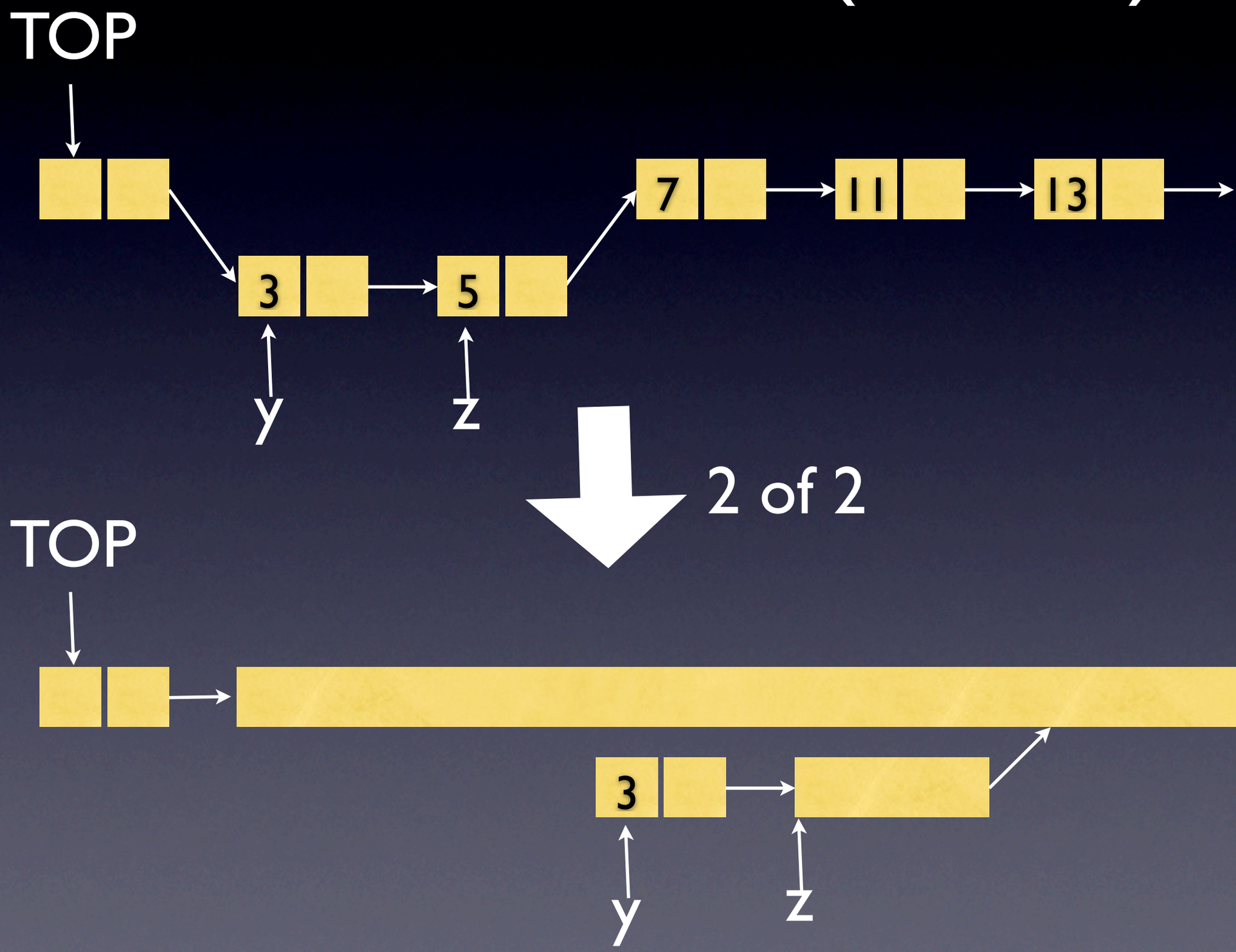


Interference (leaks)



2 of 2

Interference (leaks)



Fragment

$$
\begin{array}{rcl}
A & ::= & E = E \mid E \neq E \mid \cdots \\
S & ::= & E \mapsto E \mid \mathbf{lseg}(E, E) \mid \mathbf{tree}E \mid \cdots \\
\Pi & ::= & A_1 \wedge \cdots \wedge A_n \\
\Sigma & ::= & S_1 * \cdots * S_n \\
P, Q & ::= & \Pi \wedge \Sigma
\end{array}
$$

Shape Information Only

Symbolic Execution

$$P \xrightarrow{x:=E} x = E[x'/x] \wedge P[x'/x]$$

$$P * E{\mapsto}F \xrightarrow{x:=[E]} x = F[x'/x] \wedge (P * E{\mapsto}F)[x'/x]$$

$$P * E{\mapsto}F \xrightarrow{[E]:=F'} P * E{\mapsto}F'$$

$$P \xrightarrow{x:=\mathbf{new}()} P * x{\mapsto}-$$

$$P * E{\mapsto}F \xrightarrow{\mathbf{dispose}\ E} P$$

Safe Preconditions (re-arrangement)

Frame Inference

Frame Inference

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}$$

Frame Inference

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}$$

Frame Inference

$$P \vdash P' * F$$

Frame Inference

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}$$

Frame Inference

$$P \vdash P' * F$$

$\{\mathbf{tree}(x)\}\mathbf{dispose\_tree}(x)\{\mathbf{emp}\}$

```
...
dispose_tree(r);
dispose_tree(l);
dispose x;
...
```

Frame Inference

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}$$

Frame Inference

$$P \vdash P' * F$$

$\{\mathbf{tree}(x)\}\mathbf{dispose\_tree}(x)\{\mathbf{emp}\}$

```
...
dispose_tree(r);
dispose_tree(l);
dispose x;
...
```

$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{tree}(r)\}$

Frame Inference

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}$$

Frame Inference

$$P \vdash P' * F$$

$\{\mathbf{tree}(x)\}\mathbf{dispose\_tree}(x)\{\mathbf{emp}\}$

```
...
dispose_tree(r);
dispose_tree(l);
dispose x;
...
```

$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{tree}(r)\}$

Frame Inference

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}$$

Frame Inference

$$P \vdash P' * F$$

$\{\mathbf{tree}(x)\}\mathbf{dispose\_tree}(x)\{\mathbf{emp}\}$

...	$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{tree}(r)\}$
dispose_tree(r);	$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{emp}\}$
dispose_tree(l);	
dispose x;	
...	

Frame Inference

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}$$

Frame Inference

$$P \vdash P' * F$$

$\{\mathbf{tree}(x)\}\mathbf{dispose\_tree}(x)\{\mathbf{emp}\}$

...	$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{tree}(r)\}$
dispose_tree(r);	$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{emp}\}$
dispose_tree(l);	
dispose x;	
...	

Frame Inference

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}$$

Frame Inference

$$P \vdash P' * F$$

$\{\mathbf{tree}(x)\}\mathbf{dispose\_tree}(x)\{\mathbf{emp}\}$

...	$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{tree}(r)\}$
dispose_tree(r);	$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{emp}\}$
dispose_tree(l);	$\{x \mapsto l, r * \mathbf{emp} * \mathbf{emp}\}$
dispose x;	
...	

Frame Inference

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}$$

Frame Inference

$$P \vdash P' * F$$

$\{\mathbf{tree}(x)\}\mathbf{dispose\_tree}(x)\{\mathbf{emp}\}$

...	$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{tree}(r)\}$
dispose_tree(r);	$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{emp}\}$
dispose_tree(l);	$\{x \mapsto l, r * \mathbf{emp} * \mathbf{emp}\}$
dispose x;	
...	

Frame Inference

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}$$

Frame Inference

$$P \vdash P' * F$$

$\{\mathbf{tree}(x)\}\mathbf{dispose\_tree}(x)\{\mathbf{emp}\}$

...	$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{tree}(r)\}$
dispose_tree(r);	$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{emp}\}$
dispose_tree(l);	$\{x \mapsto l, r * \mathbf{emp} * \mathbf{emp}\}$
dispose x;	$\{\mathbf{emp} * \mathbf{emp} * \mathbf{emp}\}$
...	

Frame Inference

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}$$

Frame Inference

$$P \vdash P' * F$$

$\{\mathbf{tree}(x)\}\mathbf{dispose\_tree}(x)\{\mathbf{emp}\}$

...	$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{tree}(r)\}$
dispose_tree(r);	$\{x \mapsto l, r * \mathbf{tree}(l) * \mathbf{emp}\}$
dispose_tree(l);	$\{x \mapsto l, r * \mathbf{emp} * \mathbf{emp}\}$
dispose x;	$\{\mathbf{emp}\}$
...	

Parallel

$$\frac{\{P_1\}C_1\{Q_1\} \quad \{P_2\}C_2\{Q_2\}}{\{P_1 * P_2\}C_1 \| C_2\{Q_1 * Q_2\}}$$

Parallel rule

Parallel

$$\frac{\{P_1\}C_1\{Q_1\} \quad \{P_2\}C_2\{Q_2\}}{\{P_1 * P_2\}C_1 \| C_2\{Q_1 * Q_2\}}$$

Parallel rule

$$\frac{\{P_1\}\mathbf{proc}_1()\{Q_1\} \quad \{P_2\}\mathbf{proc}_2()\{Q_2\} \quad X \vdash P_1 * P_2 * F}{X \xrightarrow{\mathbf{proc}_1();\| \ \mathbf{proc}_2();} Q_1 * Q_2 * F}$$

Symbolic execution

Parallel

$$\frac{\{P_1\}C_1\{Q_1\} \quad \{P_2\}C_2\{Q_2\}}{\{P_1 * P_2\}C_1 \| C_2\{Q_1 * Q_2\}}$$

Parallel rule

$$\frac{\{P_1\}\mathbf{proc}_1()\{Q_1\} \quad \{P_2\}\mathbf{proc}_2()\{Q_2\} \quad X \vdash P_1 * P_2 * F}{X \xrightarrow{\mathbf{proc}_1();\| \ \mathbf{proc}_2();} Q_1 * Q_2 * F}$$

No interleavings

Symbolic execution

Atomic

$$X * S \vdash X * P * F$$

$$X * P \xrightarrow{\mathtt{C}} X'$$

$$X' \vdash Q * Y$$

$$\mathsf{stab}(Q * F) = R$$

$$X * \boxed{\mathrm{S}} \xrightarrow{\texttt{atomic C as Act;}} Y * \boxed{\mathrm{R}}$$

where Act is $P \rightsquigarrow Q$

Atomic

$$\frac{\begin{array}{l} X * S \vdash X * P * F \\ X * P \xrightarrow{\mathtt{c}} X' \\ X' \vdash Q * Y \\ \mathsf{stab}(Q * F) = R \end{array}}{X * \boxed{\mathrm{S}} \xrightarrow{\texttt{atomic C as Act;}} Y * \boxed{\mathrm{R}}}$$

Must be stable

where Act is $P \rightsquigarrow Q$

Stability Inference

Stable assertion calculation:

$$S_{n+1} = ((P \mathrel{-\!\circledast} S_n) * Q) \vee S_n$$

Stability Inference

Stable assertion calculation:

$$S_{n+1} = ((P \mathrel{-\!\circledast} S_n) * Q) \vee S_n$$

Abstract domain: [Distefano et al.]

$$S_{n+1} = \alpha((P \mathrel{-\!\circledast} S_n) * Q) \vee S_n$$

Stability Inference

Stable assertion calculation:

$$S_{n+1} = ((P \mathrel{-\!\circledast} S_n) * Q) \vee S_n$$

Abstract domain: [Distefano et al.]

$$S_{n+1} = \alpha((P \mathrel{-\!\circledast} S_n) * Q) \vee S_n$$

Septraction Elimination Procedure

Summary

Mechanisms:

- Symbolic execution
- Frame inference
- Abstraction
- Stability

Inferring actions?

Questions?

Septraction Elimination

$$(E \mapsto E') \multimap\!\circledast (F \mapsto F') \iff E{=}F \wedge E'{=}F' \wedge emp$$

$$(E \mapsto E') \multimap\!\circledast (P * Q) \iff P\restriction_E * (E \mapsto E' \multimap\!\circledast Q) \vee (E \mapsto E' \multimap\!\circledast P) * Q\restriction_E$$

$$(E \mapsto E') \multimap\!\circledast (P \vee Q) \iff (E \mapsto E' \multimap\!\circledast P) \vee (E \mapsto E' \multimap\!\circledast Q)$$

$$(P * Q) \multimap\!\circledast R \iff P \multimap\!\circledast (Q \multimap\!\circledast R)$$

$$(P \vee Q) \multimap\!\circledast R \iff (P \multimap\!\circledast R) \vee (Q \multimap\!\circledast R)$$

$$(E \mapsto E')\restriction_F \iff F \neq E * (E \mapsto E')$$

$$(P * Q)\restriction_E \iff P\restriction_E * Q\restriction_E$$

$$(P \vee Q)\restriction_E \iff P\restriction_E \vee Q\restriction_E$$