Shape Analysis of Concurrent Programs

Mooly Sagiv

. . . and also

- University of Wisconsin
  - F. DiMaio
  - D. Gopan
  - A. Loginov
  - T. Reps
- IBM Research
  - J. Field
  - H. Kolodner
  - M. Rodeh
  - E. Yahav
- Microsoft Research
  - J. Berdine
  - B. Cook
  - G. Ramalingam
- University of Massachusetts
  - N. Immerman
  - B. Hesse
- Inria
  - B. Jeannet
- Tel-Aviv University
  - D. Amit
  - I. Bogudlov
  - G. Arnold
  - G. Erez
  - N. Dor
  - T. Lev-Ami
  - R. Manevich
  - R. Shaham
  - A. Rabinovich
  - N. Rinetzky
  - G. Yorsh
  - A. Warshavsky
- Universität des Saarlandes
  - J. Bauer
  - R. Biber
  - J. Reineke
  - R. Wilhelm
- Cambridge University
  - A. Gotsman

Shape Analysis [Jones and Muchnick 1981]

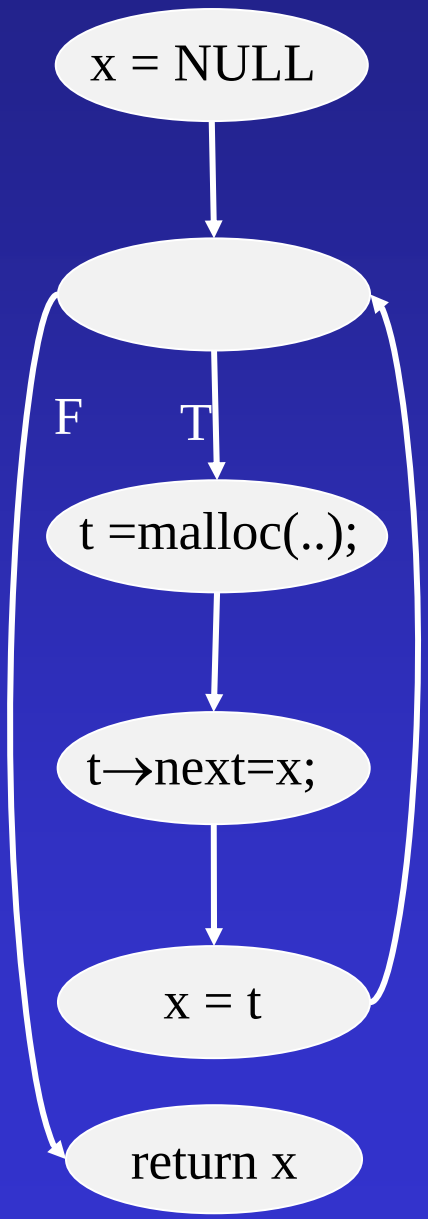
- Determine the possible shapes of a dynamically allocated data structure at a given program point
- Motivation
  - More efficient compilation
    - Compile-time garbage collection
    - Parallelization
  - Verification

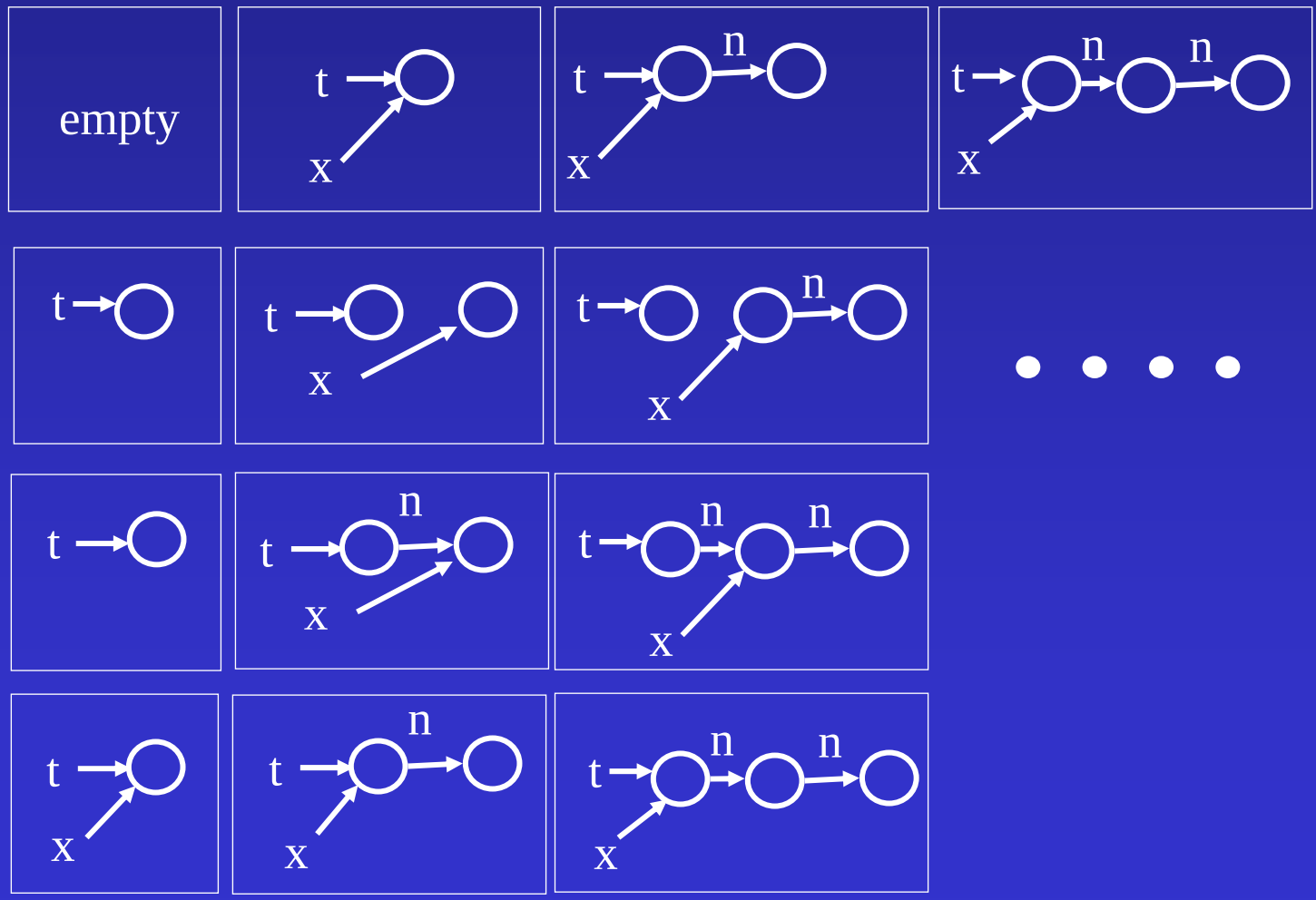
Programs and Properties

- Dynamically allocated memory
- Recursive data structures
- Recursive procedures
- Concurrency

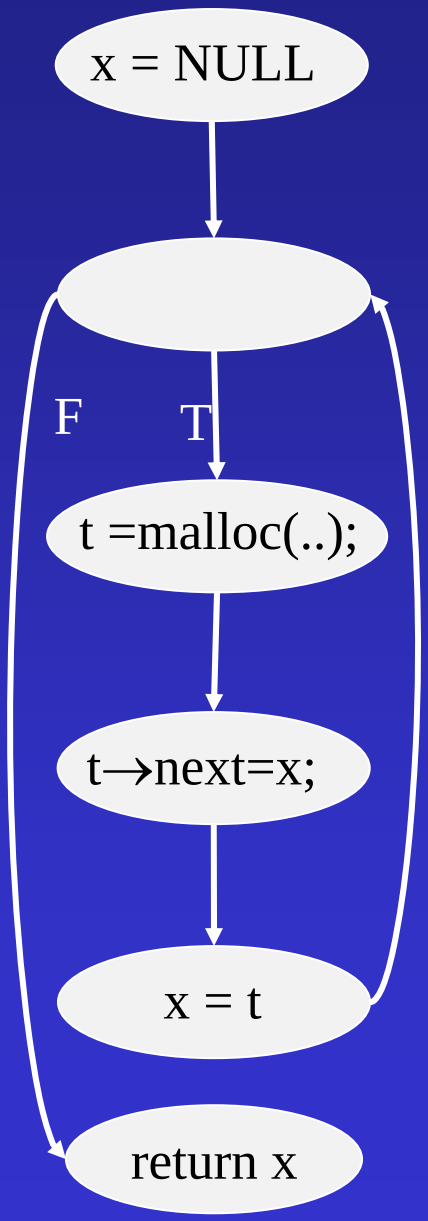
- Memory safety
- Preservation of Data structure invariants
- Partial correctness
- Linearizability
- Termination
- Lock-/Wait-freedom

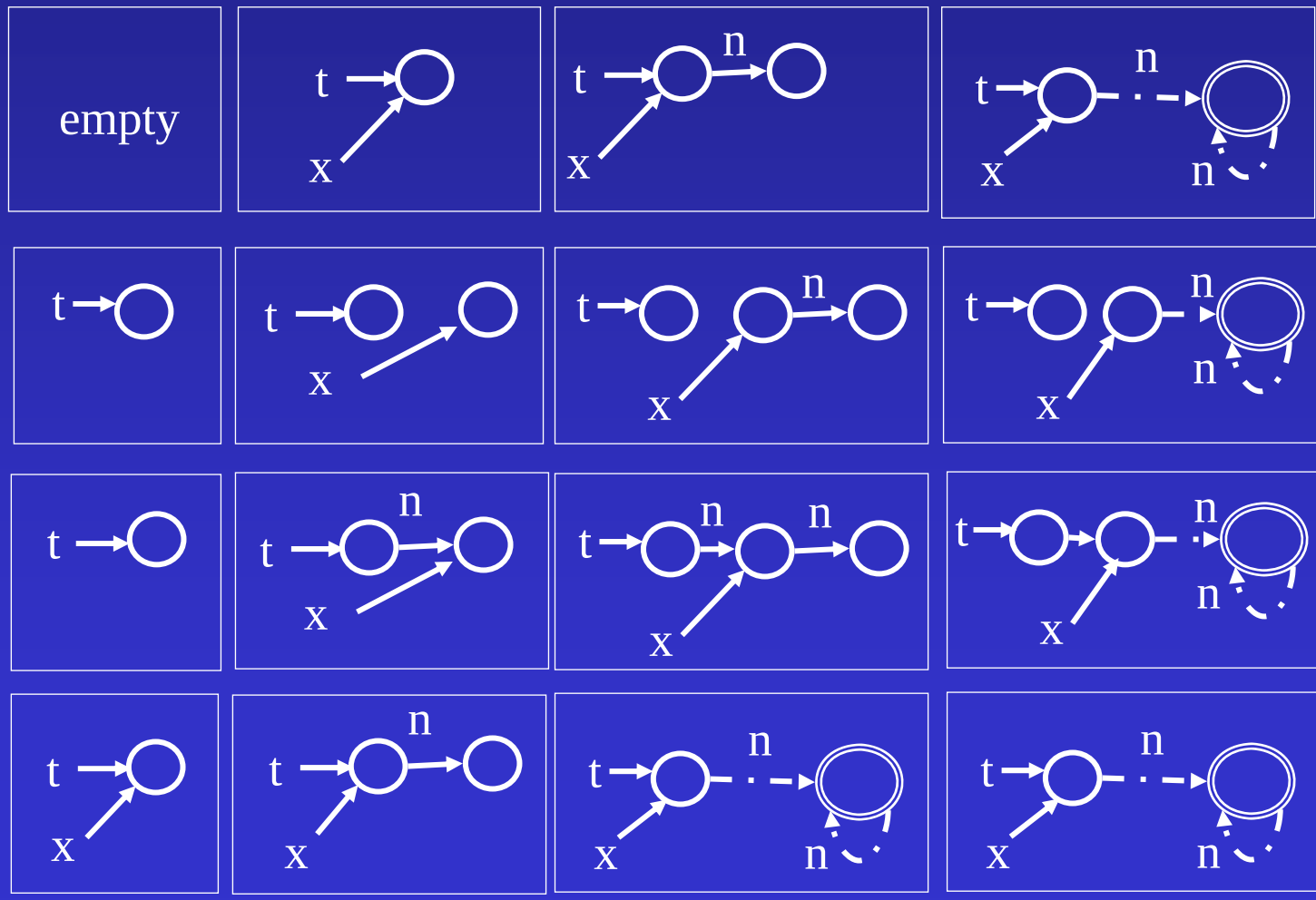
Example: Concrete Interpretation





Shape Analysis





Outline

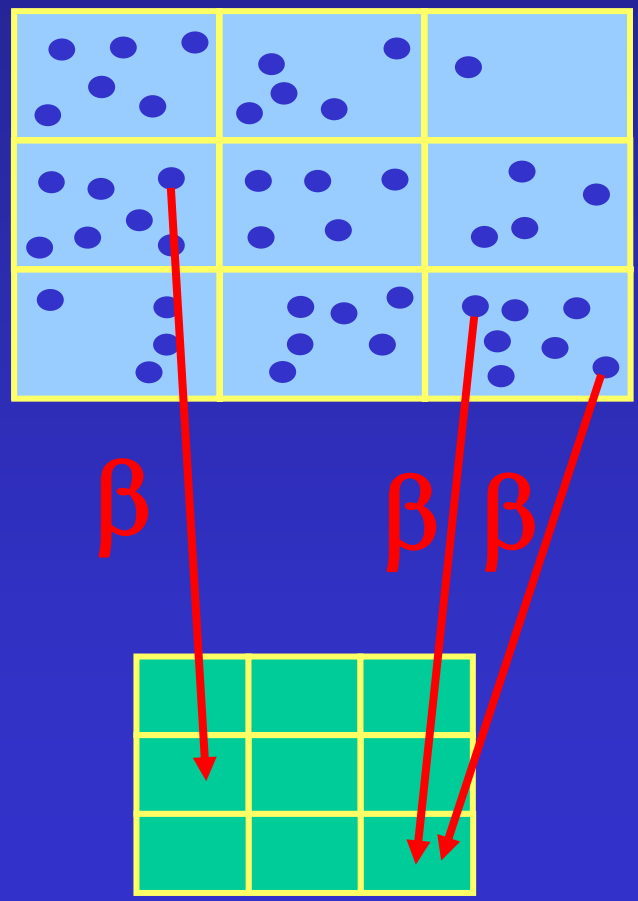
- Abstract interpretation in the nutshell
- Shape Analysis
- Handling concurrent programs

Abstract Interpretation [Cousot & Cousot]

- Checking interesting program properties is undecidable
- Use abstractions
- Every verified property holds (sound)
- But may fail to prove properties which always hold (incomplete)
  - false alarms
- Minimal false alarms

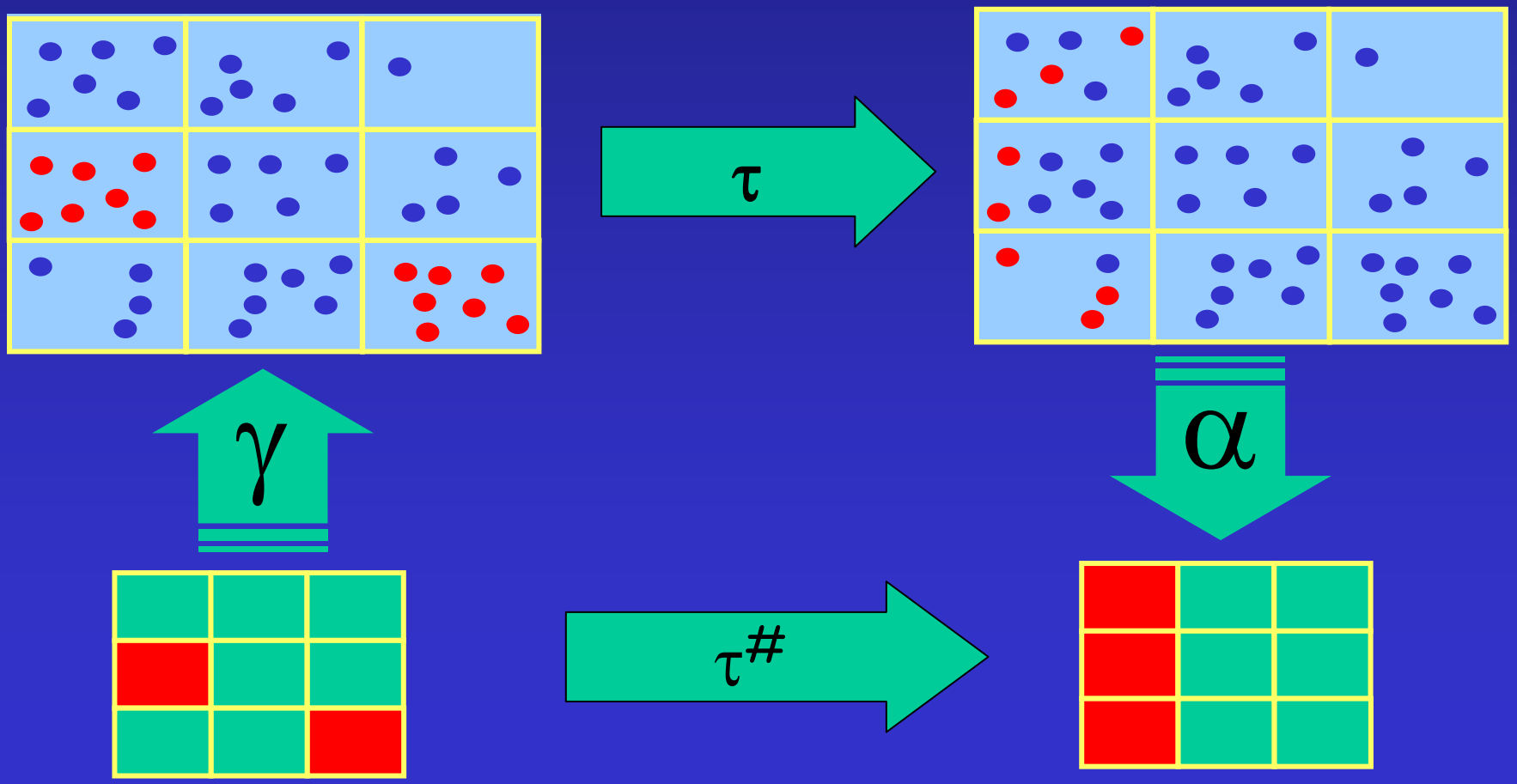
Simplified Abstract Interpretation

Concrete domain
(unbounded)



Abstract domain
(bounded)

Most Precise Abstract Transformer [Cousot, Cousot POPL 1979]

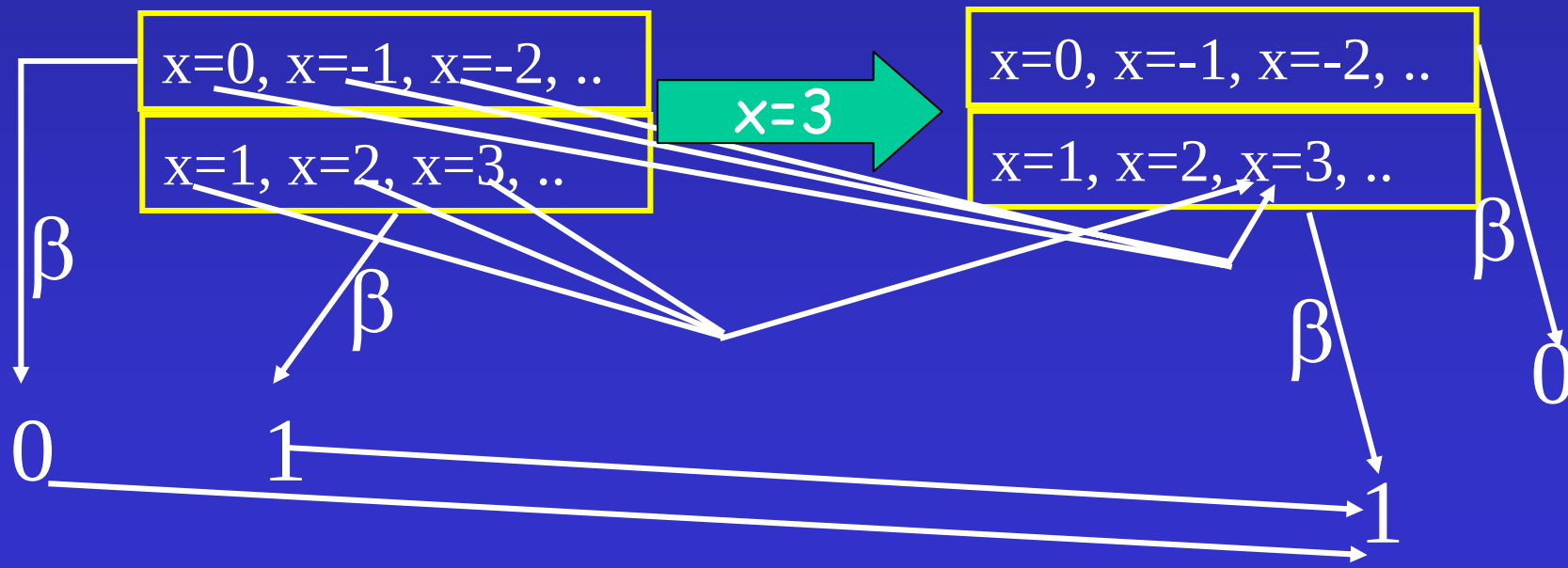


An example predicate abstraction

```
x = 3
while (true) {
   x = x +1 ;
  }
```

predicates

{x>0}

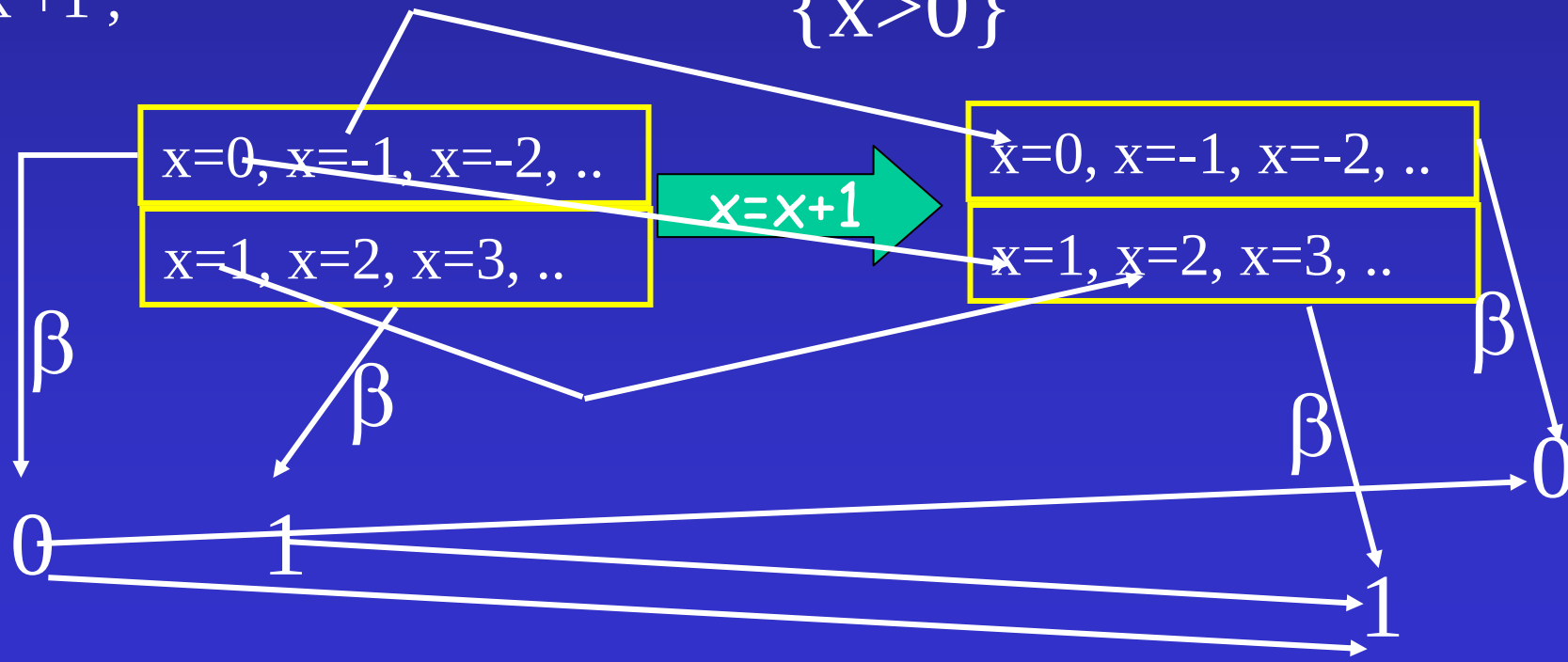


An example predicate abstraction

```
x = 3
while (true) {
   x = x +1 ;
  }
```

predicates

{x>0}



An example predicate abstraction

predicates

{x>0}

{0, 1}

x = 3

{1}

while (true) { {1}

{1} x = x +1 ; {1}

}

Shape Analysis as Abstract Interpretation

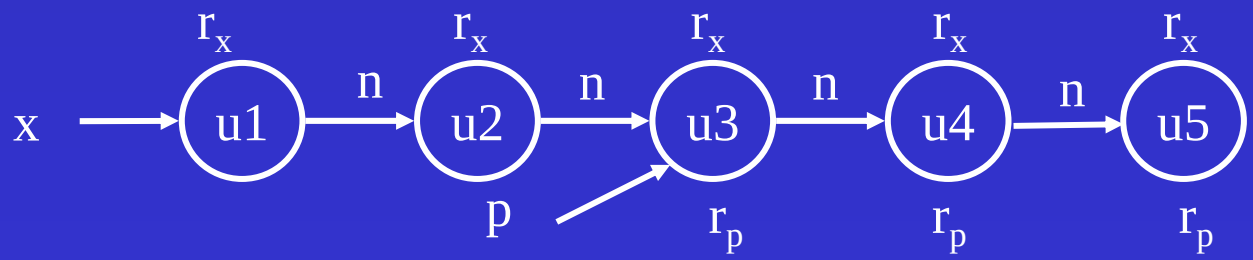
- Represent concrete stores as labeled directed graphs
  - Abstract away
    - Concrete locations
    - Primitive values
  - But unbounded
- Represent abstract stores as labeled directed graphs
  - Several concrete nodes are represented by a summary node
  - Abstract away field correlations

Representing Concrete Stores by Logical Structures

- Parametric vocabulary
- Heap
  - Locations ≈ Individuals
  - Program variables ≈ Unary relations
  - Fields ≈ Binary relations

Representing Concrete Stores by Logical Structures

- U = {u1, u2, u3, u4, u5}
- x = {u1}, p = {u3}
- n = {<u1, u2>, <u2, u3>, <u3, u4>, <u4, u5>}
- r_x = {u1, u2, u3, u4, u5}
- r_p = {u3, u4, u5}



Representing Abstract Stores by 3-Valued Logical Structures

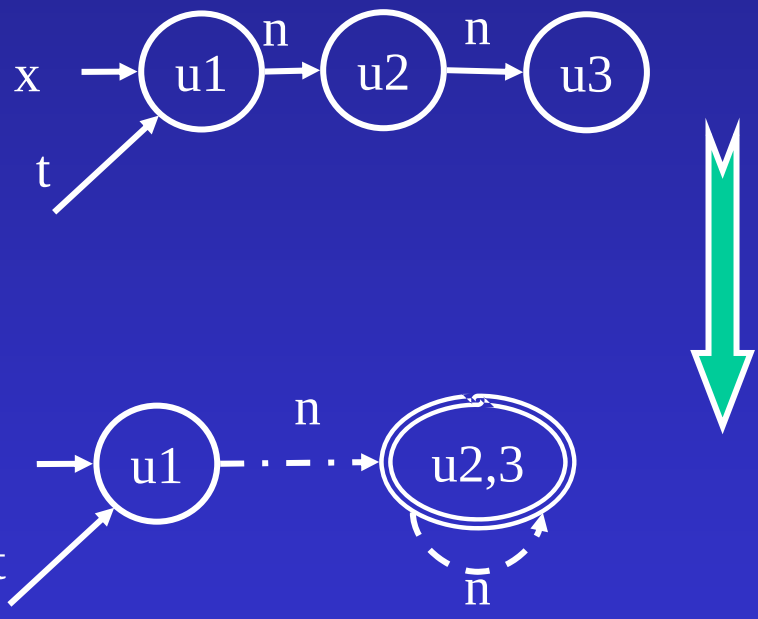
- A join semi-lattice: $0 \sqcup 1 = 1/2$
- {0, 1, ½} values for relations

Canonical Abstraction (β)

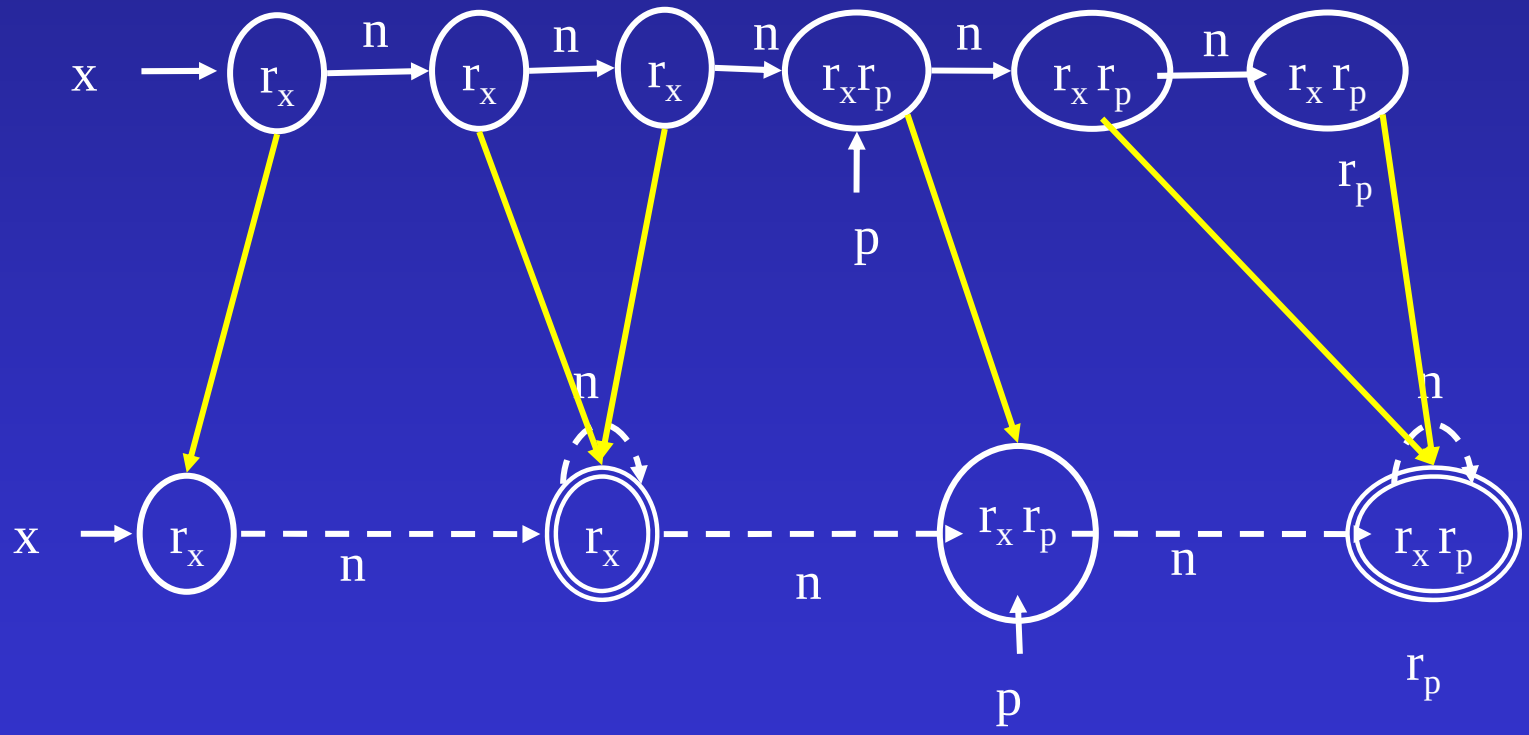
- Partition the individuals into equivalence classes based on the values of their unary relations
  - Every individual is mapped into its equivalence class
- Collapse binary relations via ⊔
  - $p^S(u'_1, u'_2) = \sqcup \{p^B(u_1, u_2) \mid f(u_1)=u'_1,\ f(u_2)=u'_2\}$
- At most 2^A abstract individuals

Canonical Abstraction (β)

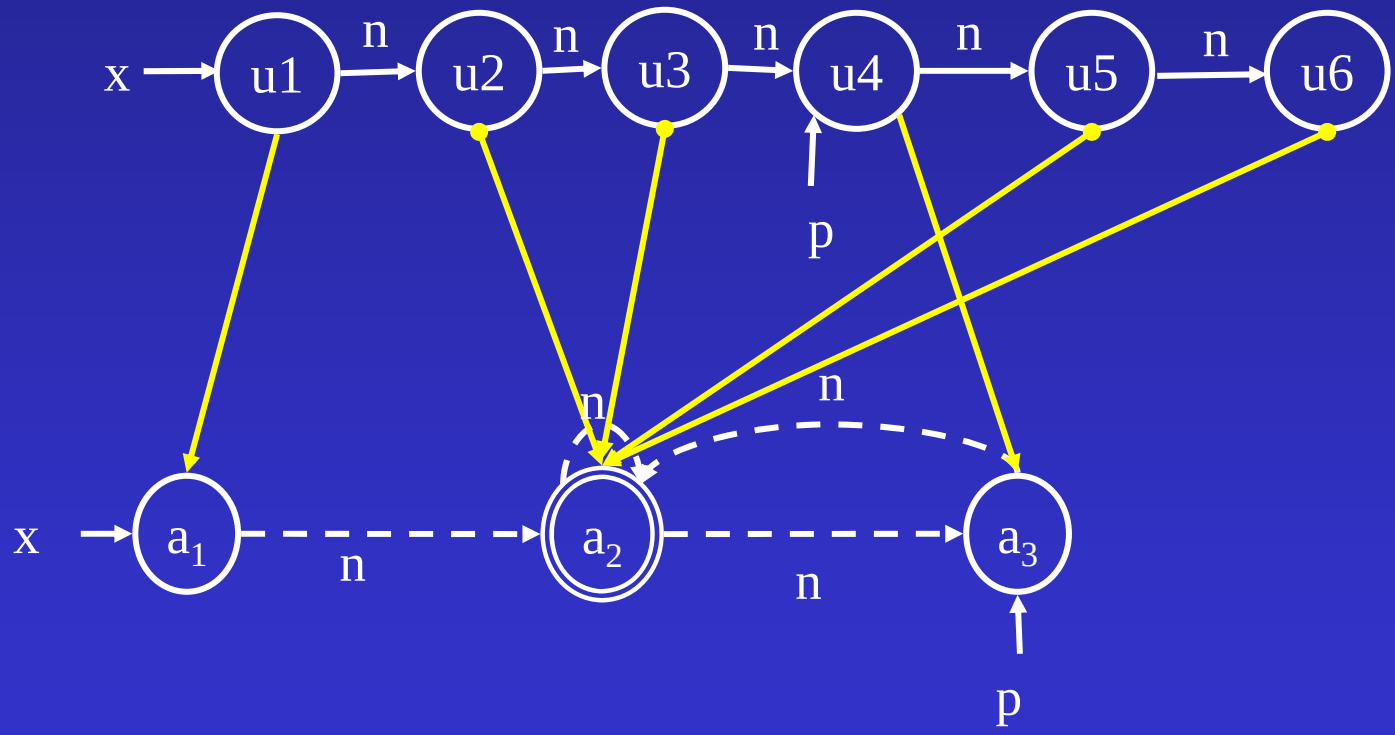
```
x = NULL;
while (…) do {
      t = malloc();
      t →next=x;
     x = t
}
```



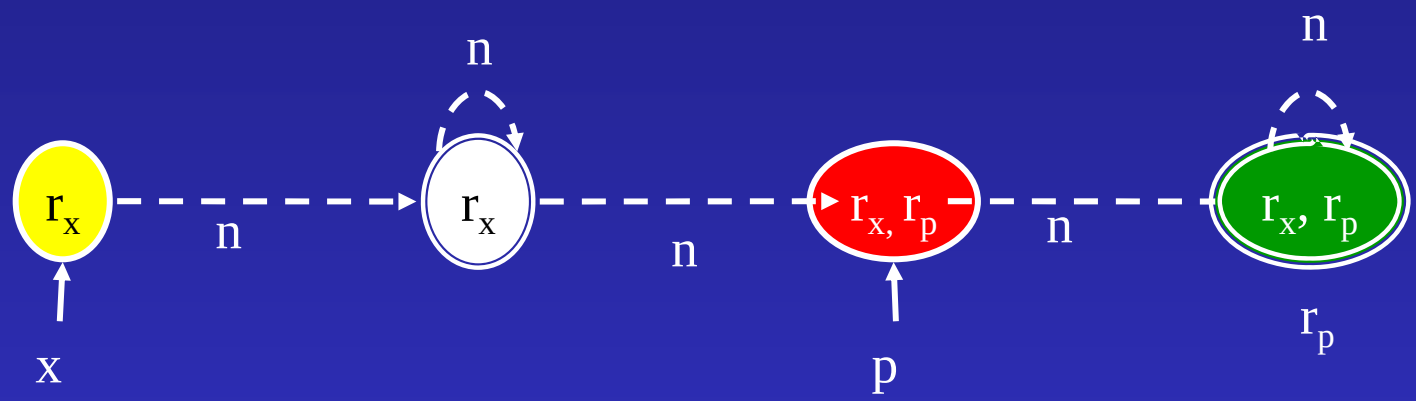
Canonical Abstraction



Canonical Abstraction w/o reachability



Canonical Abstractions as Formulas [Yorsh'03, Kuncak'04, Wies'07]

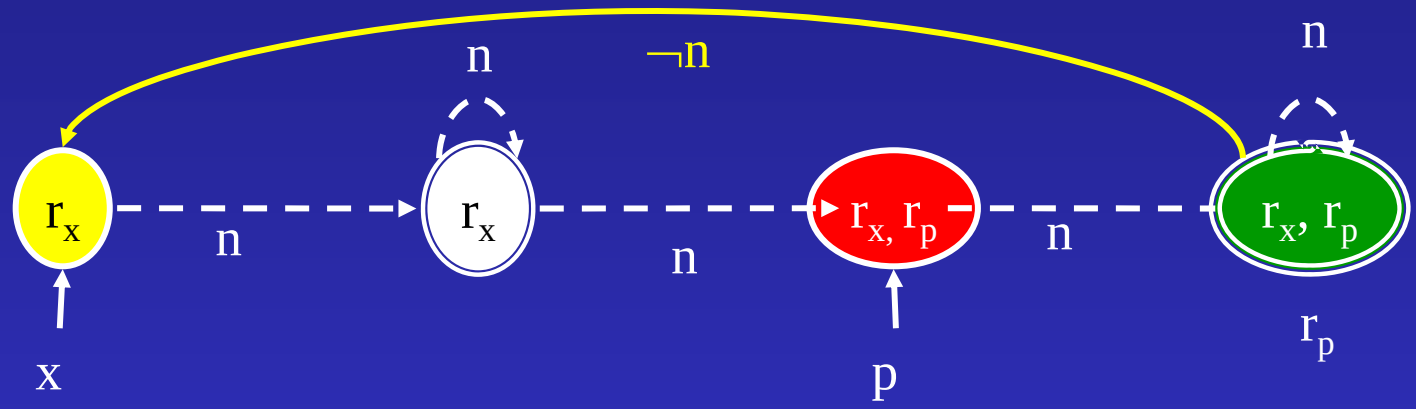


$\forall v: (x(v) \wedge r_x(v) \wedge \neg p(v) \wedge \neg r_p(v)) \vee$
$(\neg x(v) \wedge r_x(v) \wedge \neg p(x) \wedge \neg r_p(v)) \vee$
$(\neg x(v) \wedge r_x(v) \wedge p(v) \wedge r_p(v)) \vee$
$(\neg x(v) \wedge r_x(v) \wedge \neg p(v) \wedge r_p(v)))$

$\forall v: r_x(v) \Leftrightarrow \exists w: x(w) \wedge n^*(w, v)$

$\forall v: r_p(v) \Leftrightarrow \exists w: p(w) \wedge n^*(w, v)$

Canonical Abstractions as Formulas [Yorsh'03, Kuncak'04, Wies'07]



$$\forall v, w: (\neg x(v) \wedge r_x(v) \wedge \neg p(v) \wedge r_p(v)) \wedge (x(w) \wedge r_x(w) \wedge \neg p(w) \wedge \neg r_p(w))$$

$$\Rightarrow$$

$$\neg n(v, w)$$

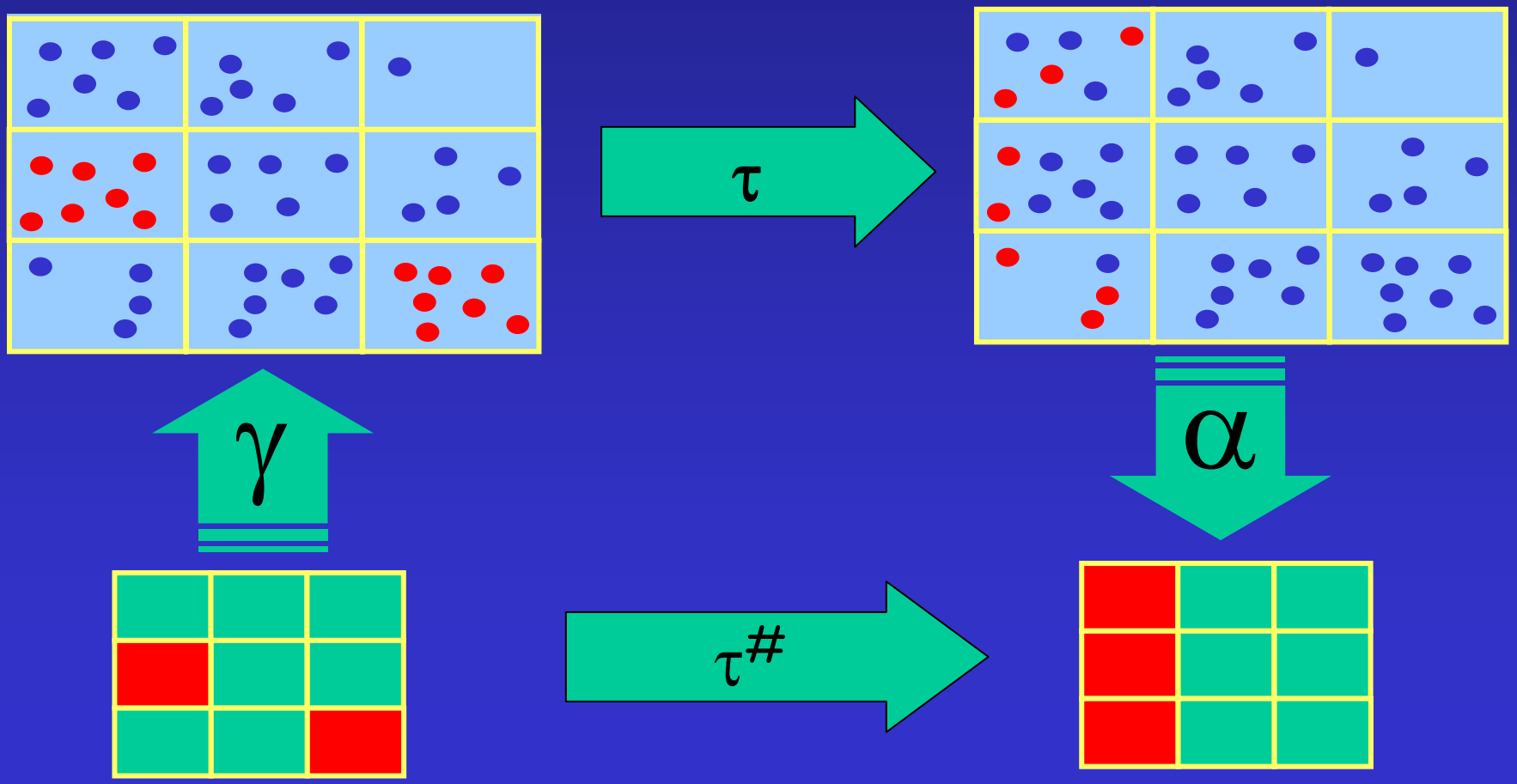
Canonical Abstraction

- Limited form of quantified invariants

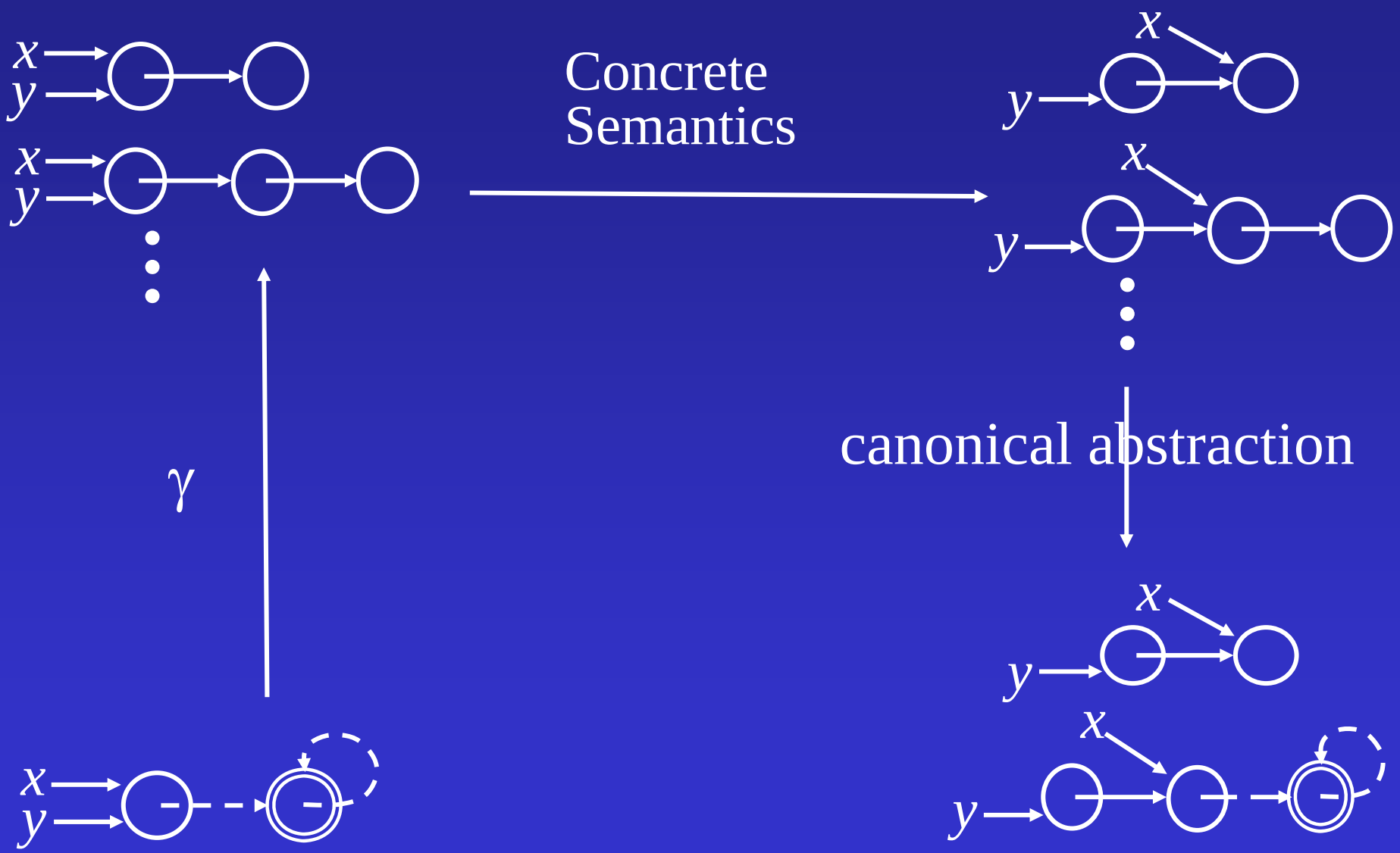
$$\left\{ \begin{array}{l} \vee i \\ \forall v: \vee j \ \{\varphi_{i,j}(v)\} \\ \wedge \\ \forall v, w: \wedge m, n \ \{\varphi_{i,m}(v) \wedge \varphi_{i,n}(w) \Rightarrow \varepsilon_{i,m,n}(v, w)\} \end{array} \right\}$$

  - quantifier alternation and transitive closure only in instrumentation
  - Bounded number of distinctions

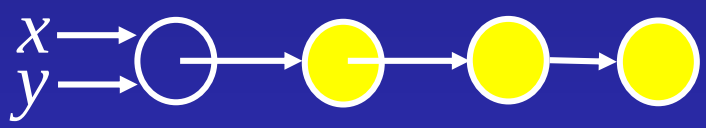
Most Precise Abstract Transformer [Cousot, Cousot POPL 1979]

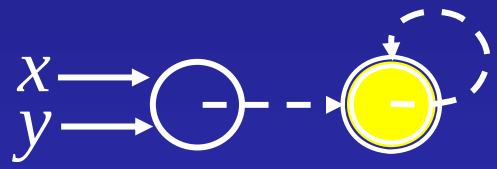


Best Transformer ($x = x \rightarrow n$)

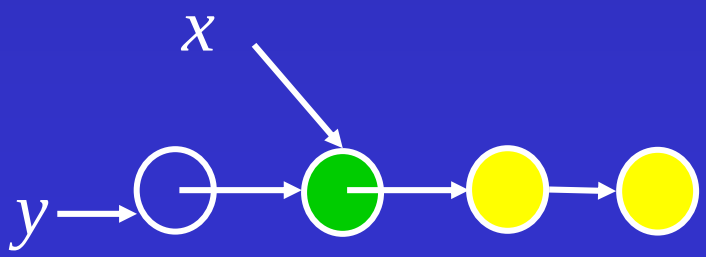


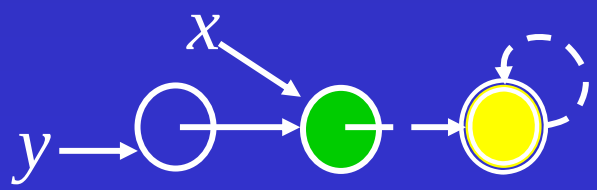
Non-Fixed-Partition





x = x→n

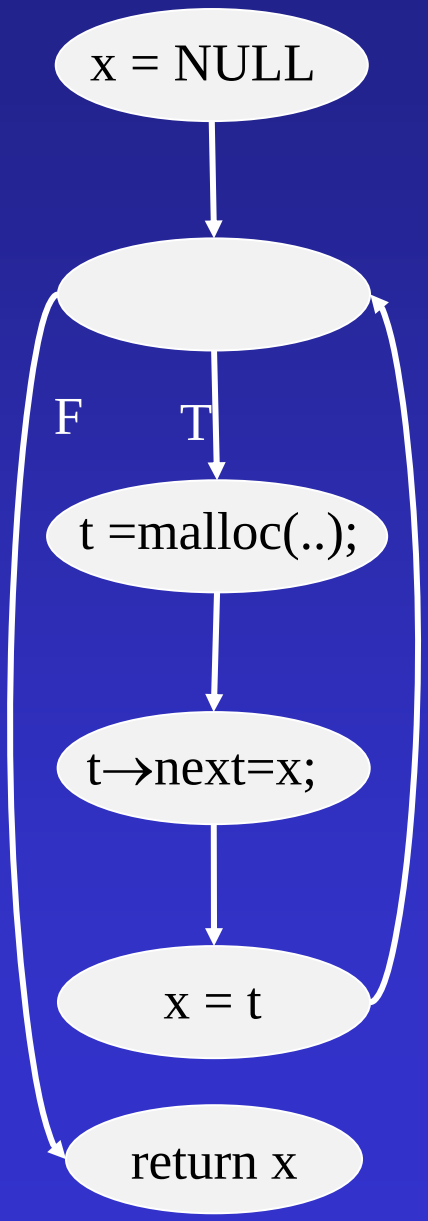


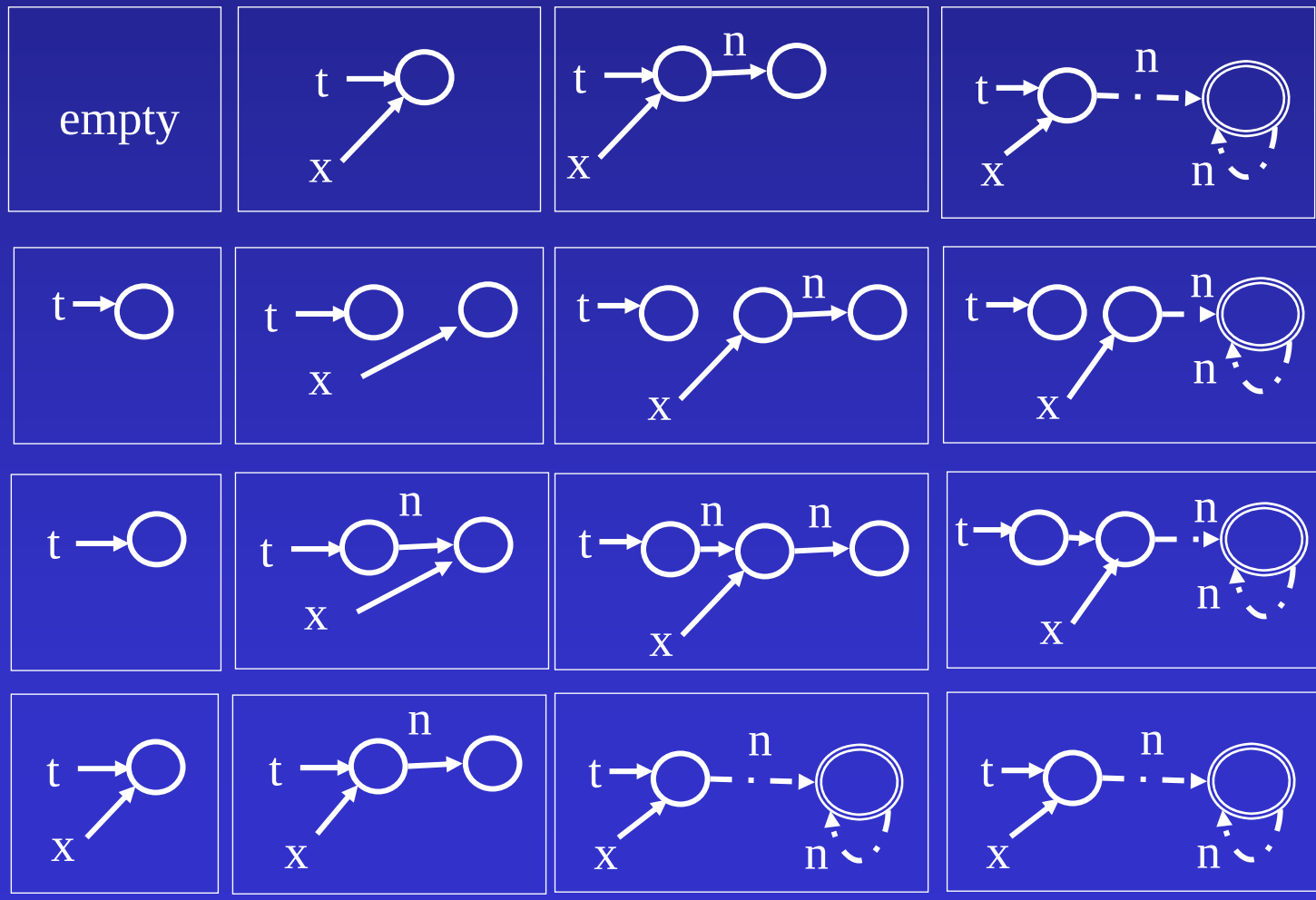


Canonical Abstraction

- Limited form of quantified invariants
  - quantifier alternation only in instrumentation
- Not a static memory partition
  - The same memory location can be represented by different abstract nodes in different shape graphs

Shape Analysis





[TOPLAS'02, Lev-Ami, SAS'00]

- Concrete transformers using first order formulas
- Effective algorithms for computing transformers
  - Partial concretization
  - 3-valued logic Kleene evaluation
  - Finite differencing & incremental algorithms [Reps, ESOP'03]
- A parametric yacc like system[TVLA]
  - http://www.cs.tau.ac.il/~tvla

Challenges in shape analysis

- Programming language features
  - Procedures
  - Modularity & encapsulation
  - Concurrency
- Properties
- Complex data structures
  - Hierarchy
  - Mixture of data and heap
  - Array of heaps
- Scaling to larger programs

Concurrency

- Models threads as ordinary objects [Yahav, POPL'01]
- Thread-modular shape analysis [Gotsman, PLDI'07]
- Heap decomposition [Manevich, SAS'08]
- Thread quantification [Berdine, CAV'08]
- Enforcing a locking regime [Rinetzkey]

Thread Quantification for Concurrent Shape Analysis

J. Berdine, T. Lev-Ami, R. Manevich, G. Ramalingam, and M. Sagiv

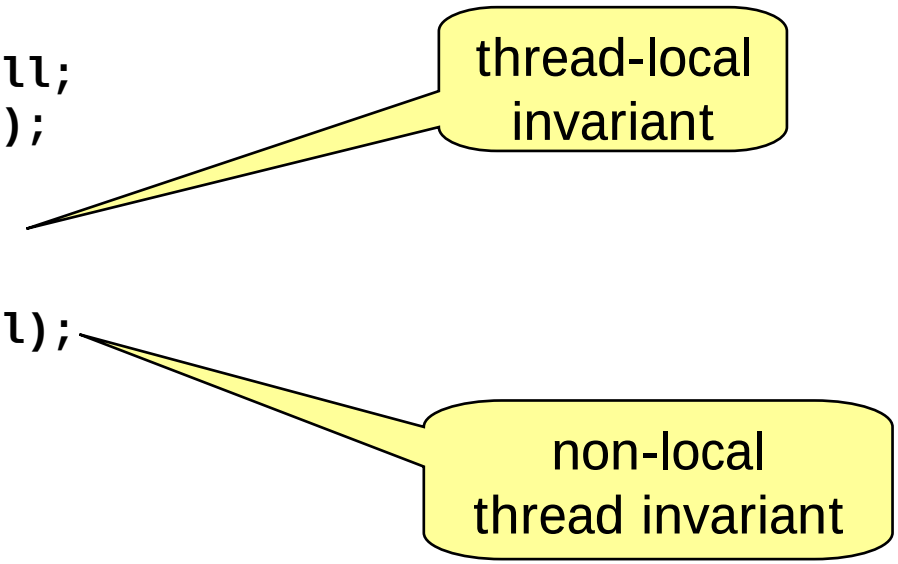
CAV'08

Non-blocking stack [Treiber 1986]

```
[1] void push(Stack *S, data_type v) {
[2]     Node *x = alloc(sizeof(Node));
[3]     x->d = v;
[4]     do {
[5]         Node *t = S->Top;
[6]         x->n = t;
[7]     } while (!CAS(&S->Top,t,x));
[8] }


[9]  data_type pop(Stack *S){
[10]     do {
[11]         Node *t = S->Top;
[12]         if (t == NULL)
[13]             return EMPTY;
[14]         Node *s = t->n;
[15]         data_type r = s->d;
[16]         } while (!CAS(&S->Top,t,s));
[17]     return r;
[18] }
```

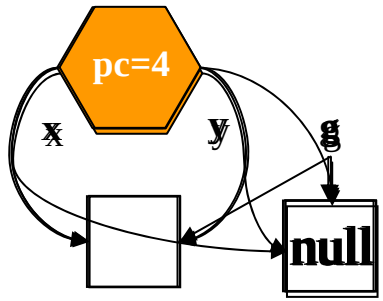
Toy example

```
Object g = null; // global variable


threadProc() {
  Object x=null, y=null;
  [1] x = new Object();
  [2] y = x;
  [3] assert(x == y);
      g = x;
  [4] assert(g != null);
   // assert(g = x);
}
```



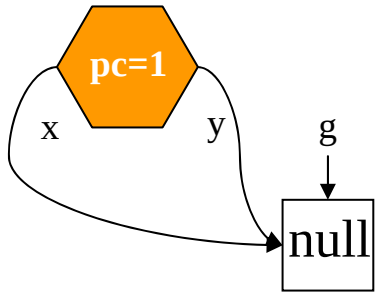
Toy example (one thread)

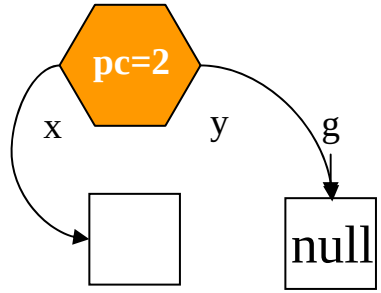
```
Object g = null; // global variable


threadProc() {
  Object x=null, y=null;
  [1] x = new Object();
  [2] y = x;
  [3] assert(x == y);
      g = x;
  [4] assert(g != null);
   // assert(g = x);
}
```

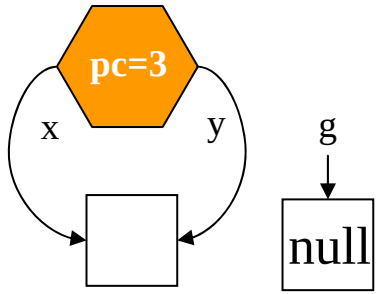


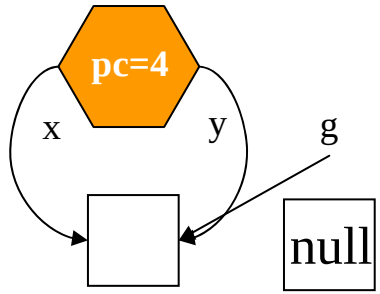
Toy example (one thread)

```
Object g = null; // global variable


threadProc() {
  Object x=null, y=null;
  [1] x = new Object();
  [2] y = x;
  [3] assert(x == y);
      g = x;
  [4] assert(g != null);
   // assert(g = x);
}
```









Toy example (one thread)

```
Object g = null; // global variable


threadProc() {
  Object x=null, y=null;
  [1] x = new Object();
  [2] y = x;
  [3] assert(x == y);
      g = x;
  [4] assert(g != null);
   // assert(g = x);
}
```

$$pc = 1 \wedge x = y = g = null$$

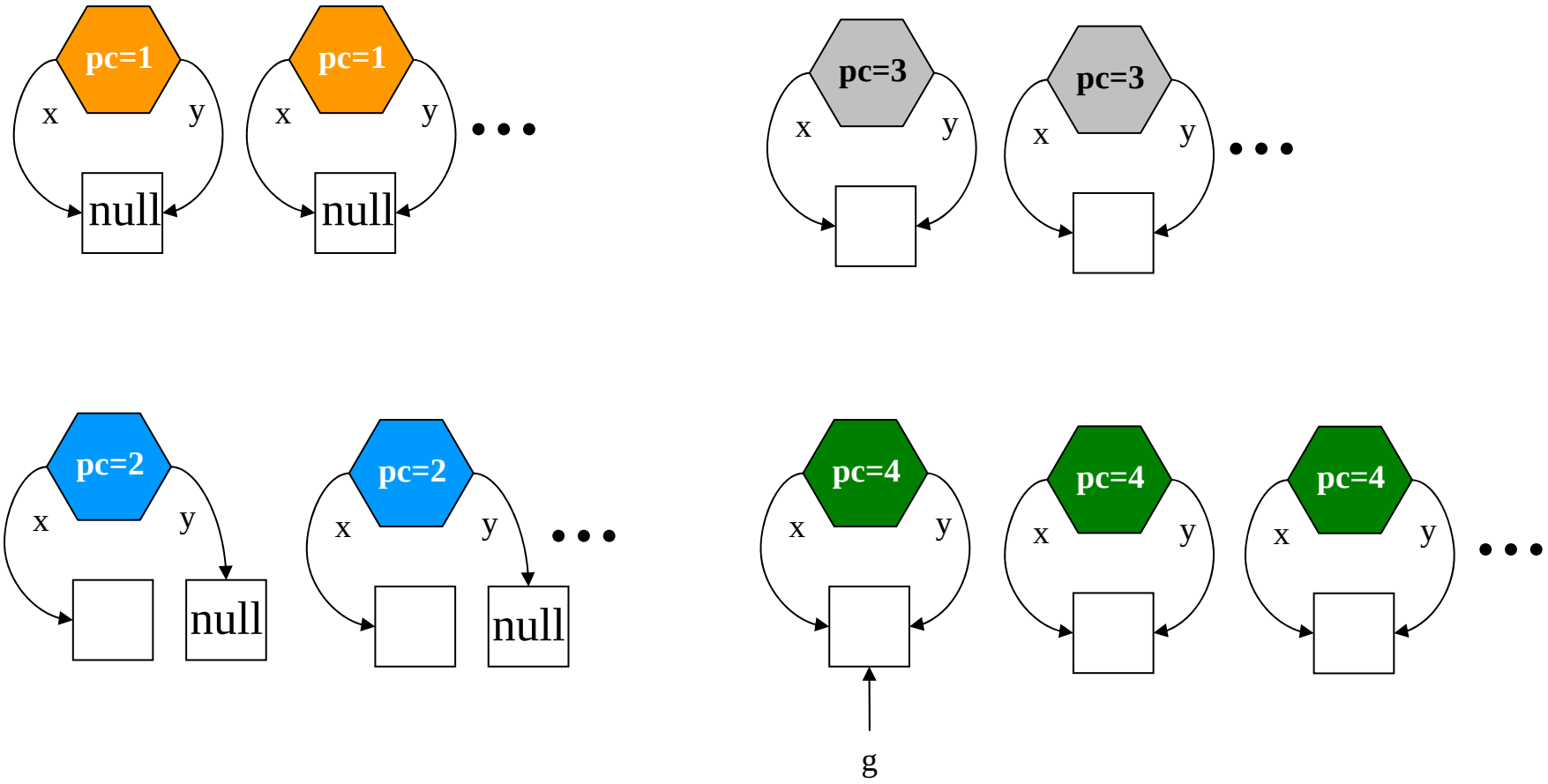
$$\vee\ pc = 2 \wedge x \neq null \wedge y = g = null$$

$$\vee\ pc = 3 \wedge x = y \neq null \wedge g = null$$

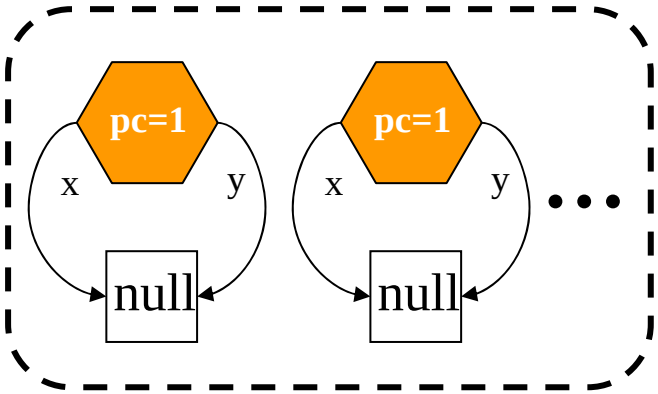
$$\vee\ pc = 4 \wedge x = y = g \wedge g \neq null$$

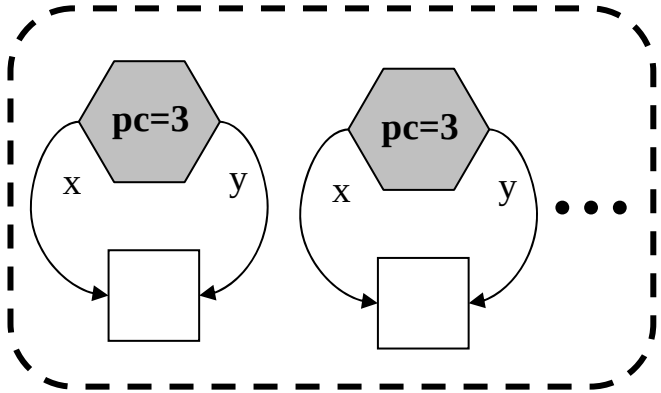
What about an unbounded number of threads?

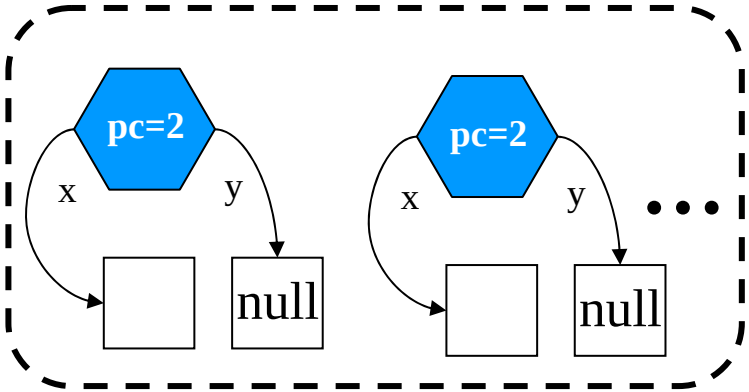
Multiple threads

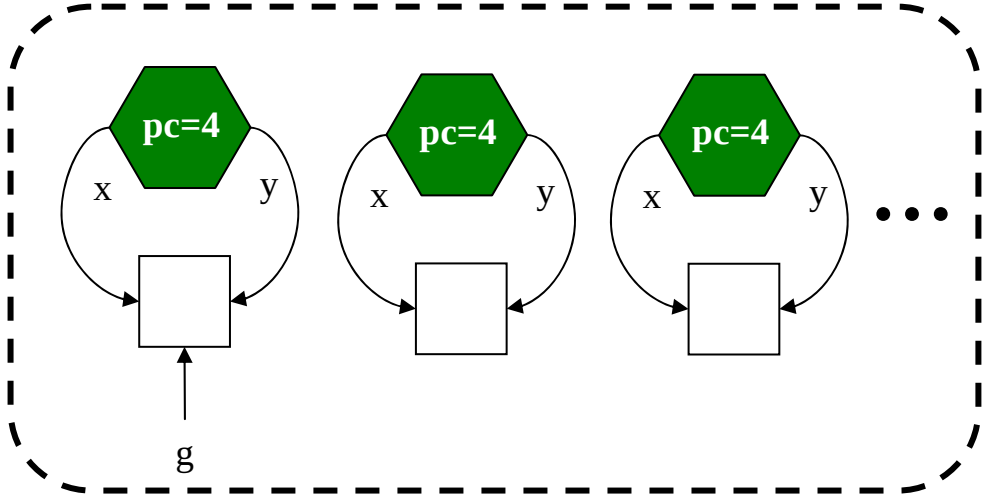


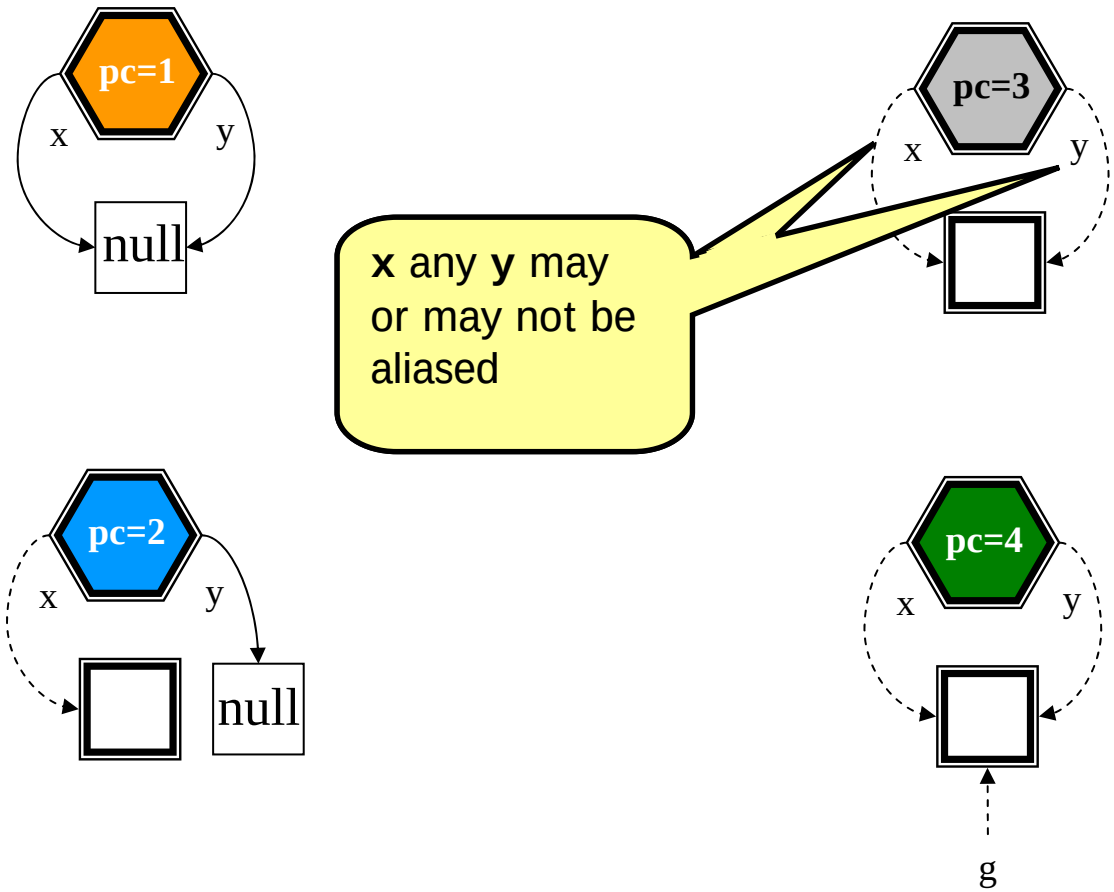
Collapse threads at the same location
Yahav, POPL'01












```
threadProc() {
  Object x=null, y=null;
  [1] x = new Object();
  [2] y = x;
  [3] assert(x == y);
      g = x;
  [4] assert(g != null);
   // assert(g = x);
}
```

Lost correlations between local variables of same thread

Quantified Abstraction

```
Object g = null; // global variable

threadProc() {
  Object x=null, y=null;
  [1] x = new Object();
  [2] y = x;
  [3] assert(x == y);
      g = x;
  [4] assert(g != null);
   // assert(g = x);
}
```

$\forall t$

$pc(t) = 1 \wedge \ldots$

$\vee\ pc(t) = 2 \wedge \ldots$

$\vee\ pc(t) = 3 \wedge \mathbf{x(t) = y(t)} \ \ldots$

$\vee\ pc(t) = 4 \wedge x(t) = y(t) \wedge \mathbf{g \neq null}$

$\vee\ pc(t) = 4 \wedge x(t) = y(t) = g \wedge \mathbf{g \neq null}$

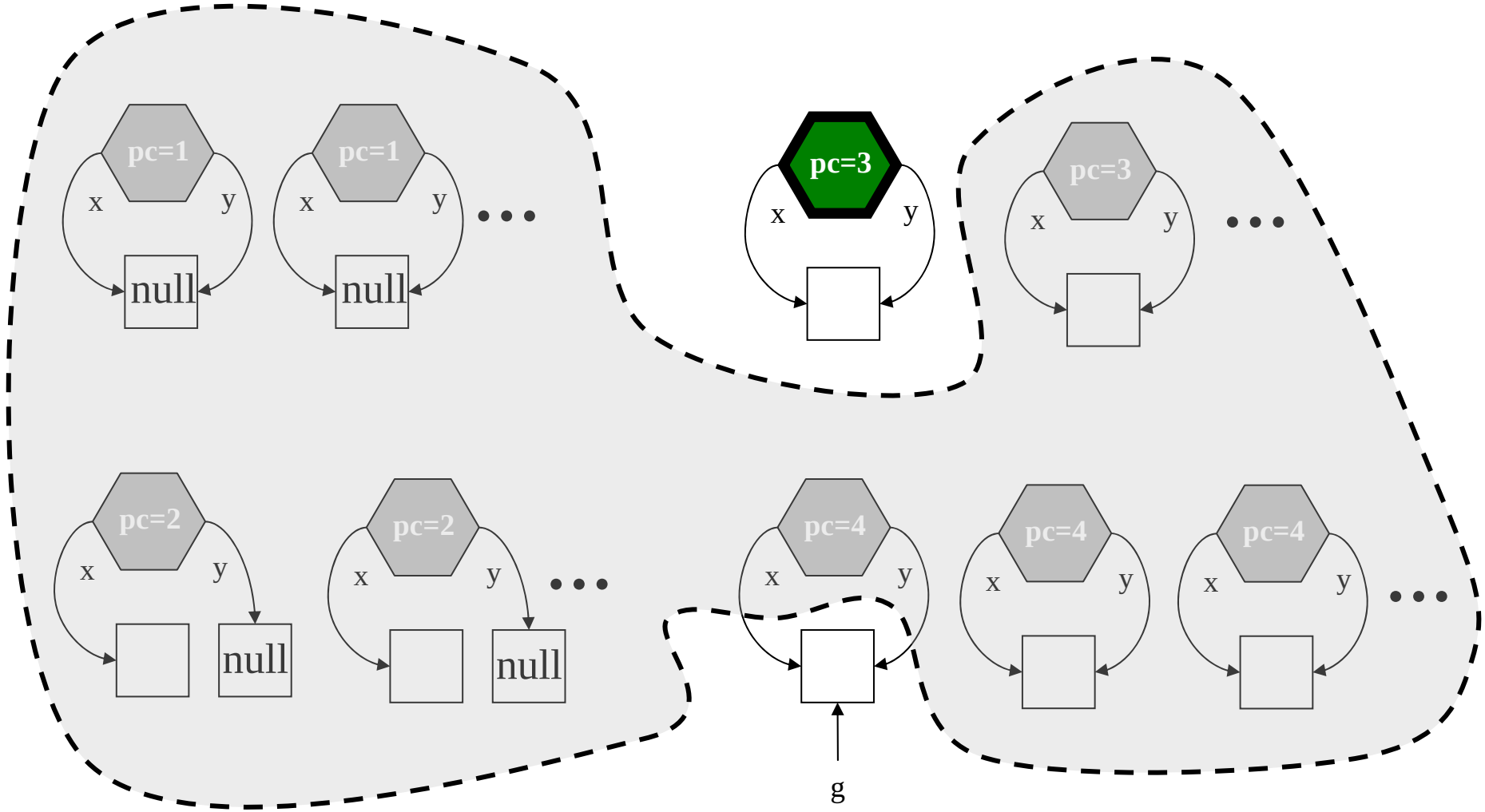
$\ldots$

Each disjunct represents an abstraction of state from the perspective of one thread

Lost information: number of threads

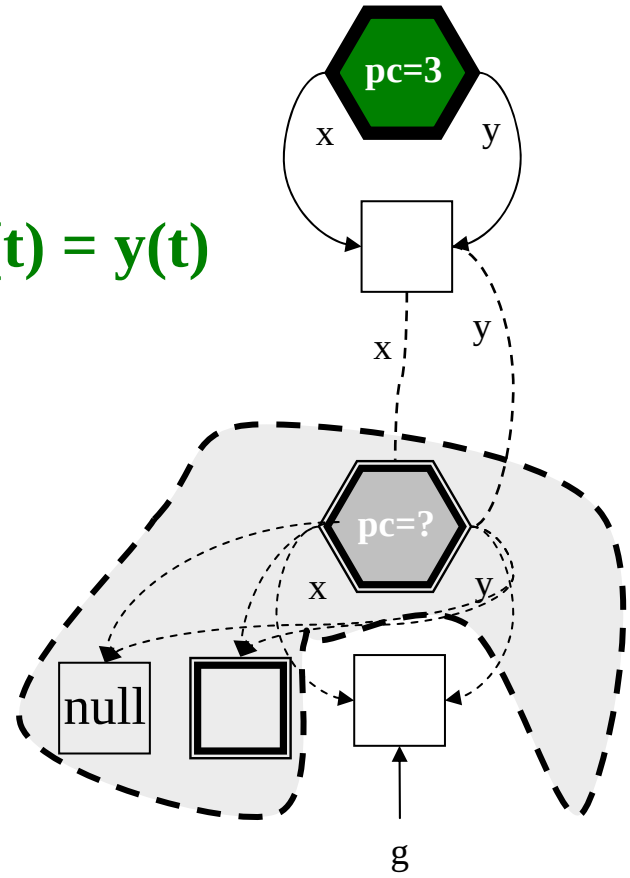
Indexed predicate

Intuition for quantified abstraction



Intuition for quantified abstraction

$$\forall t \quad \begin{array}{l} \ldots \\ \vee\ pc(t) = 3 \wedge \mathbf{x(t) = y(t)} \\ \ldots \end{array}$$



Information maintained

- (dis)equalities between local variables of each thread and global variables (and null)

Information lost

- How many threads are there?
- (dis)equalities between variables of different threads

Properties of quantified abstraction

- Abstracts the number of threads
  - (Similar to counter abstraction with ≥ 0)
- Thread-modular abstraction
  - Each disjunction focuses on a single thread
  - Coarse abstraction of “environment”
  - Captures correlations between values of local variables of a thread and the global state
  - Abstracts away correlations between the values local variables of different threads

Canonical Heaps

$\vee i$:

$\forall v: \vee j \{ \quad \varphi_{i,j}(v) \}$

$\wedge$

$\forall\, v, w: \wedge m, n \{ \quad \varphi_{i,m}(v) \wedge \varphi_{i,n}(w) \Rightarrow \varepsilon_{i,m,n}(v, w) \}$

“Lifted” Canonical Heaps

$$\forall t: \quad \vee i: \quad \left\{ \begin{array}{l} \forall v: \vee j \ \{\varphi_{i,j}(t, v)\} \\ \wedge \\ \forall v, w: \wedge m, n\{ \ \varphi_{i,m}(t, v) \wedge \varphi_{i,n}(t, w) \Rightarrow \varepsilon_{i,m,n}(t, v, w) \ \} \end{array} \right\}$$

Missing

- Heuristics for computing transformers
  - Quantifier instantiation
- Proving linearizability
  - [Amit, CAV'07]
  - Fixed linearization point
  - Bounded concrete differences between sequential and concurrent implementarions
  - Simplified memory model
    - Garbage collection
    - Sequential consistency

Non-blocking stack [Treiber 1986]

```
[1] void push(Stack *S, data_type v) {
[2]     Node *x = alloc(sizeof(Node));
[3]     x->d = v;
[4]     do {
[5]         Node *t = S->Top;
[6]         x->n = t;
[7]     } while (!CAS(&S->Top,t,x));
[8] }


[9]  data_type pop(Stack *S){
[10]     do {
[11]         Node *t = S->Top;
[12]         if (t == NULL)
[13]             return EMPTY;
[14]         Node *s = t->n;
[15]         data_type r = s->d;
[16]         } while (!CAS(&S->Top,t,s));
[17]     return r;
[18] }
```

Experimental results

Verified Programs	**#states**	**time (sec.)**
Treiber's stack [1986]	764	7
Two-lock queue [Doherty and Groves FORTE'04]	3,415	17
Non-blocking queue [Michael and Scott PODC'96]	10,333	252

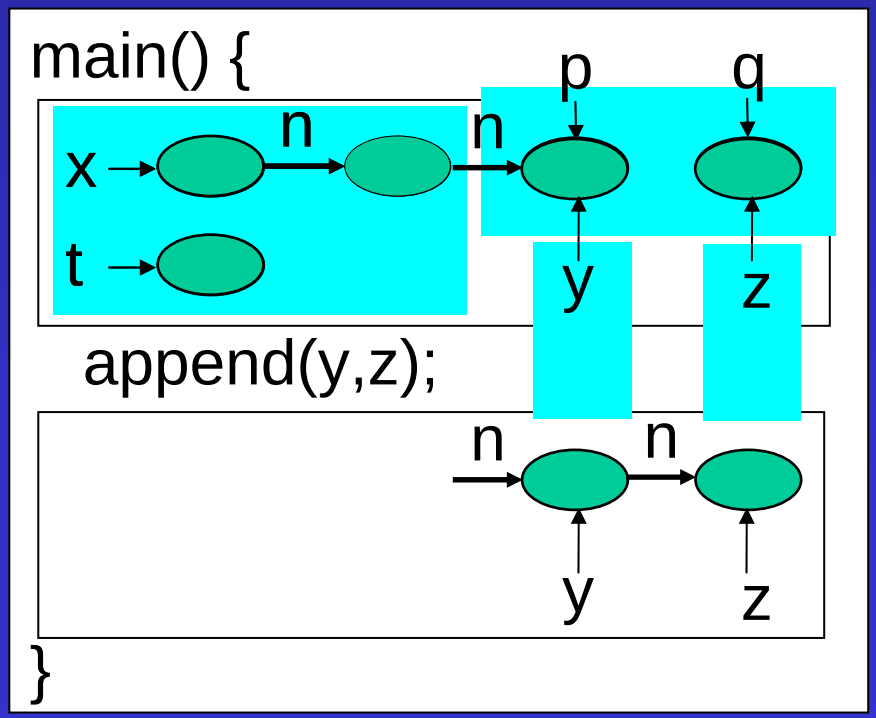
First automatic verification of linearizability for unbounded number of threads

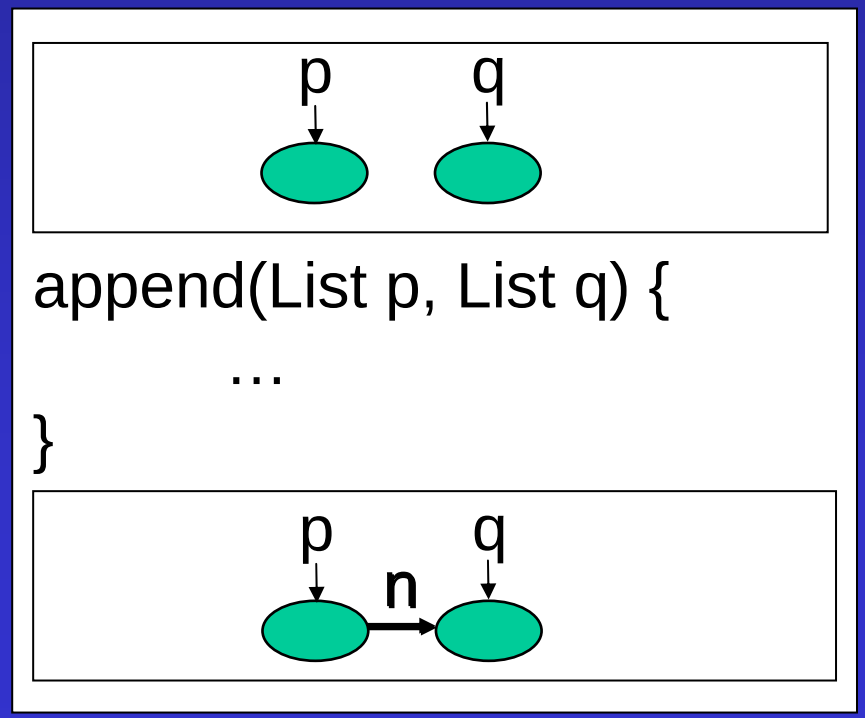
Summary

- Shape analysis is an interesting abstract interpretation problem
- Limited forms of quantified invariants can be utilized to prove interesting properties
- Scaling shape analysis to realistic programs is still an open problem

How to tabulate procedures?

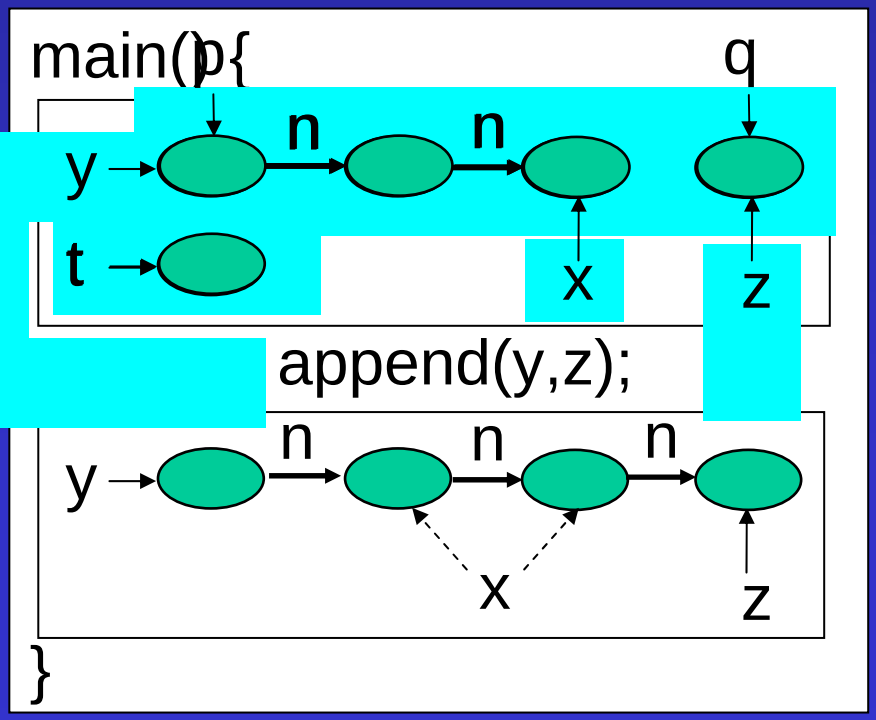
- Procedure ≡ input/output relation
  - Not reachable ➔ Not effected
  - proc: local (≡reachable) heap ➔ local heap

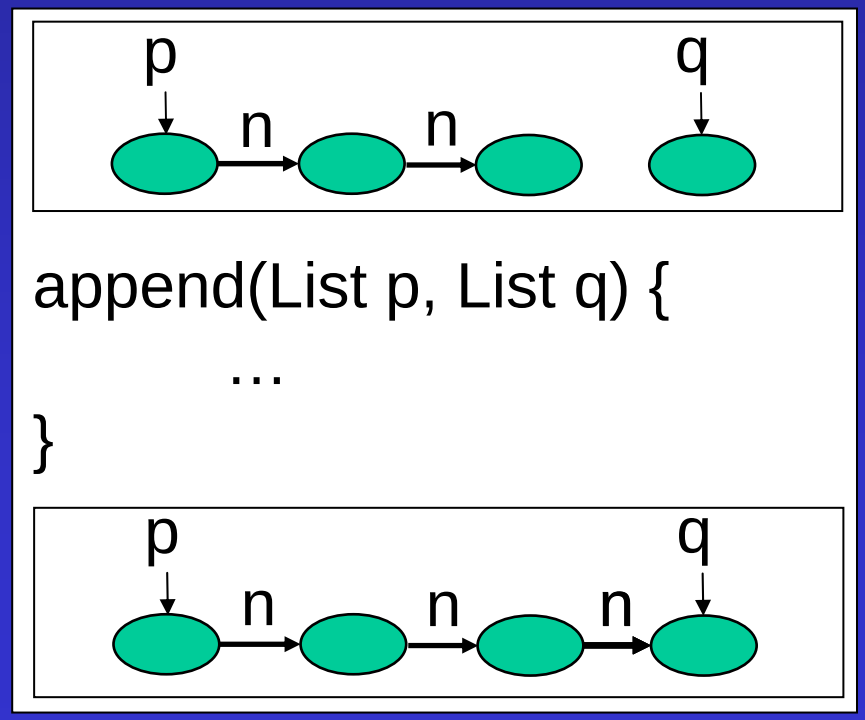




How to handle sharing?

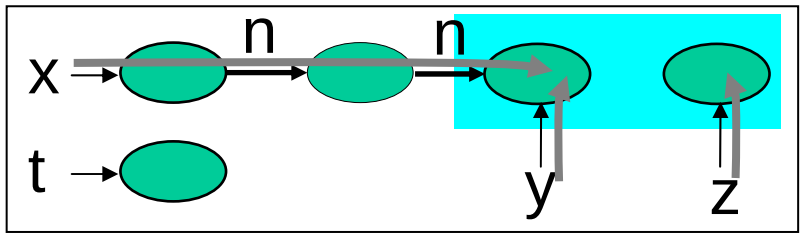
- External sharing may break the functional view





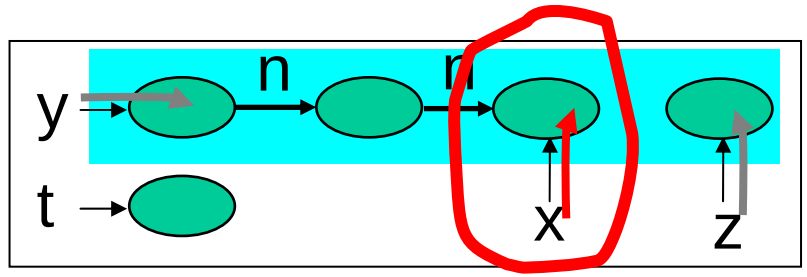
What's the difference?

1st Example



append(y,z);

2nd Example

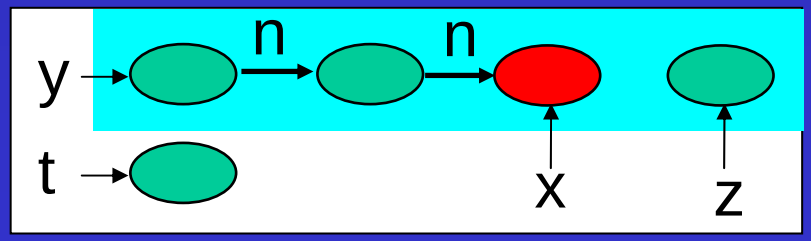


append(y,z);

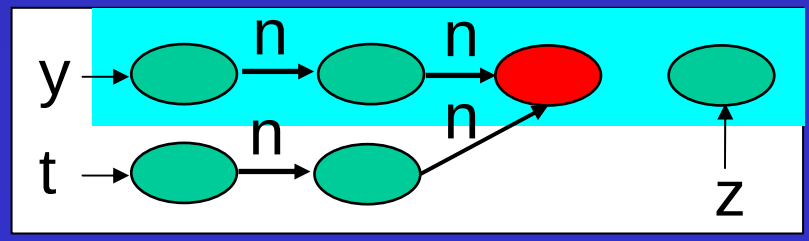
Cutpoints

- An object is a **cutpoint** for an invocation
  - Reachable from actual parameters
  - Not pointed to by an actual parameter
  - Reachable without going through a parameter

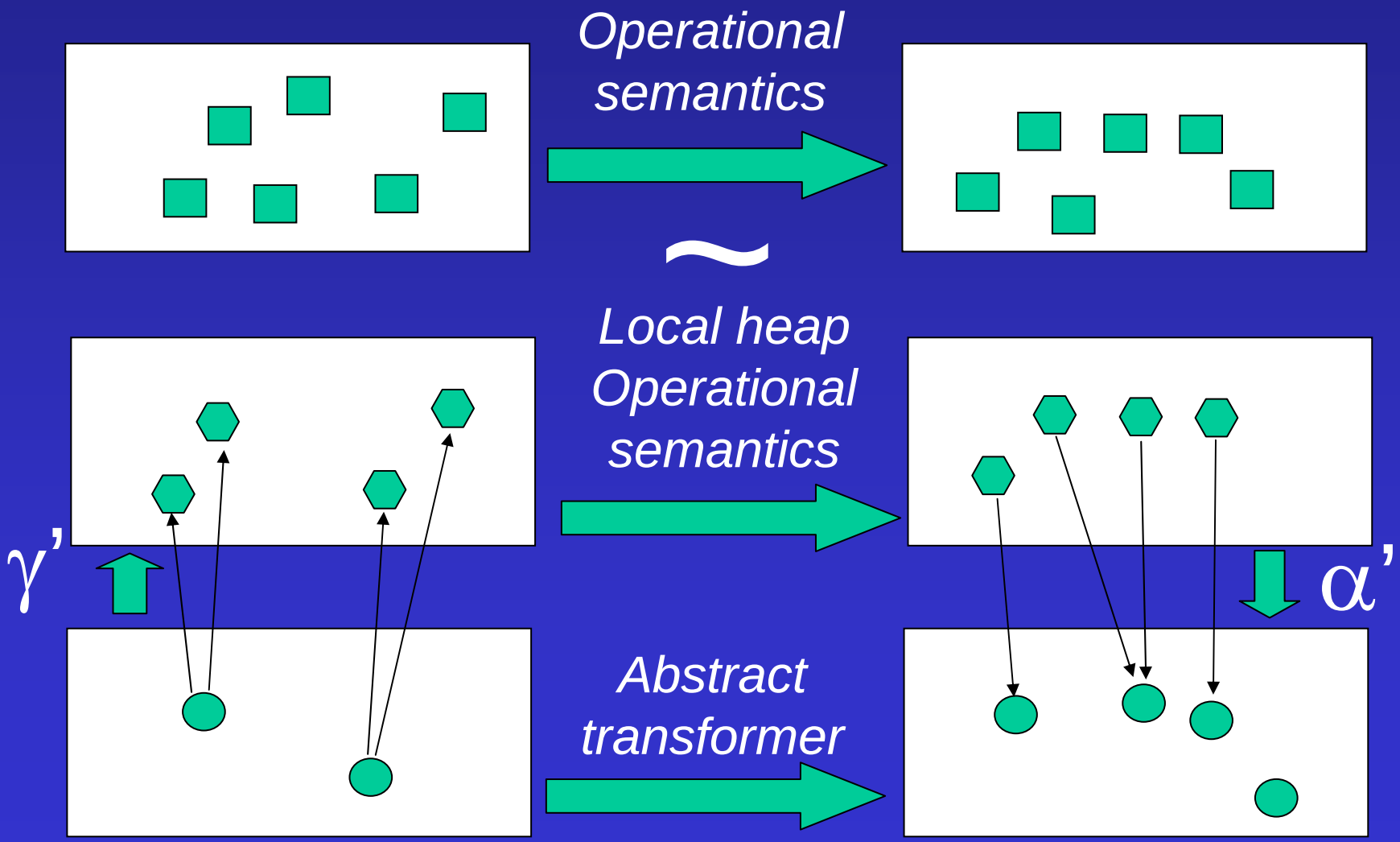
append(y,z)



append(y,z)

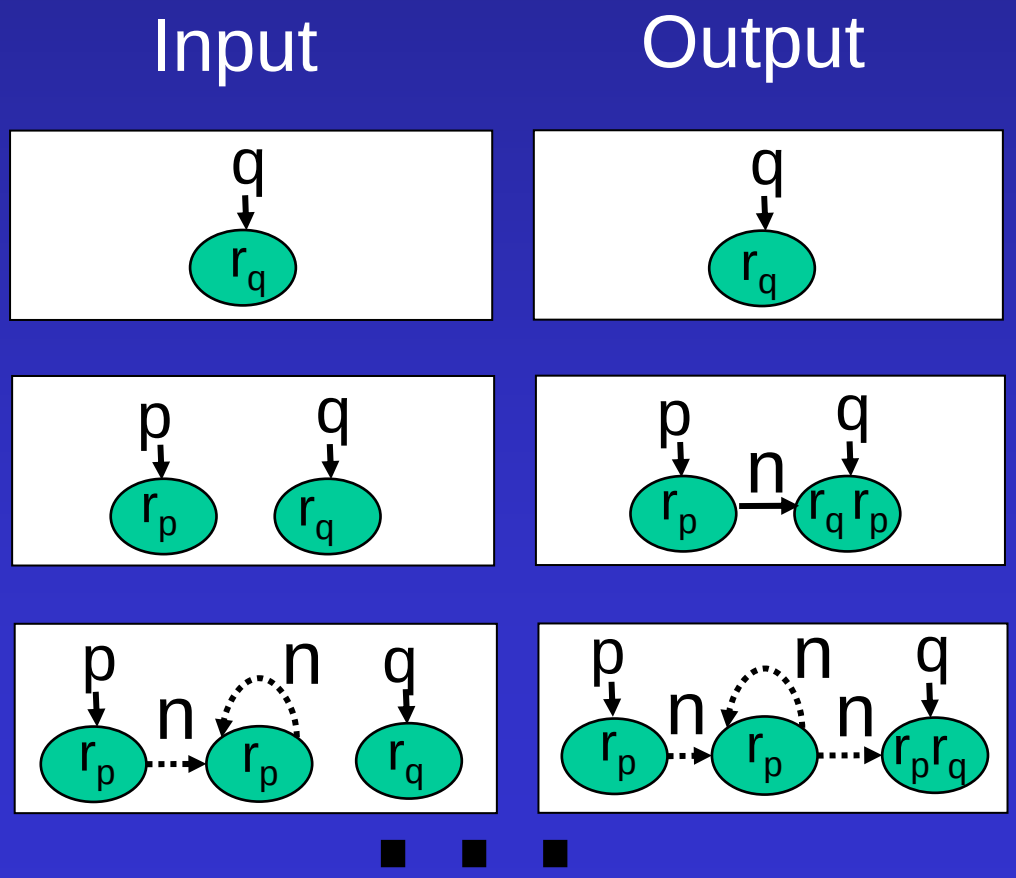


Introducing local heap semantics



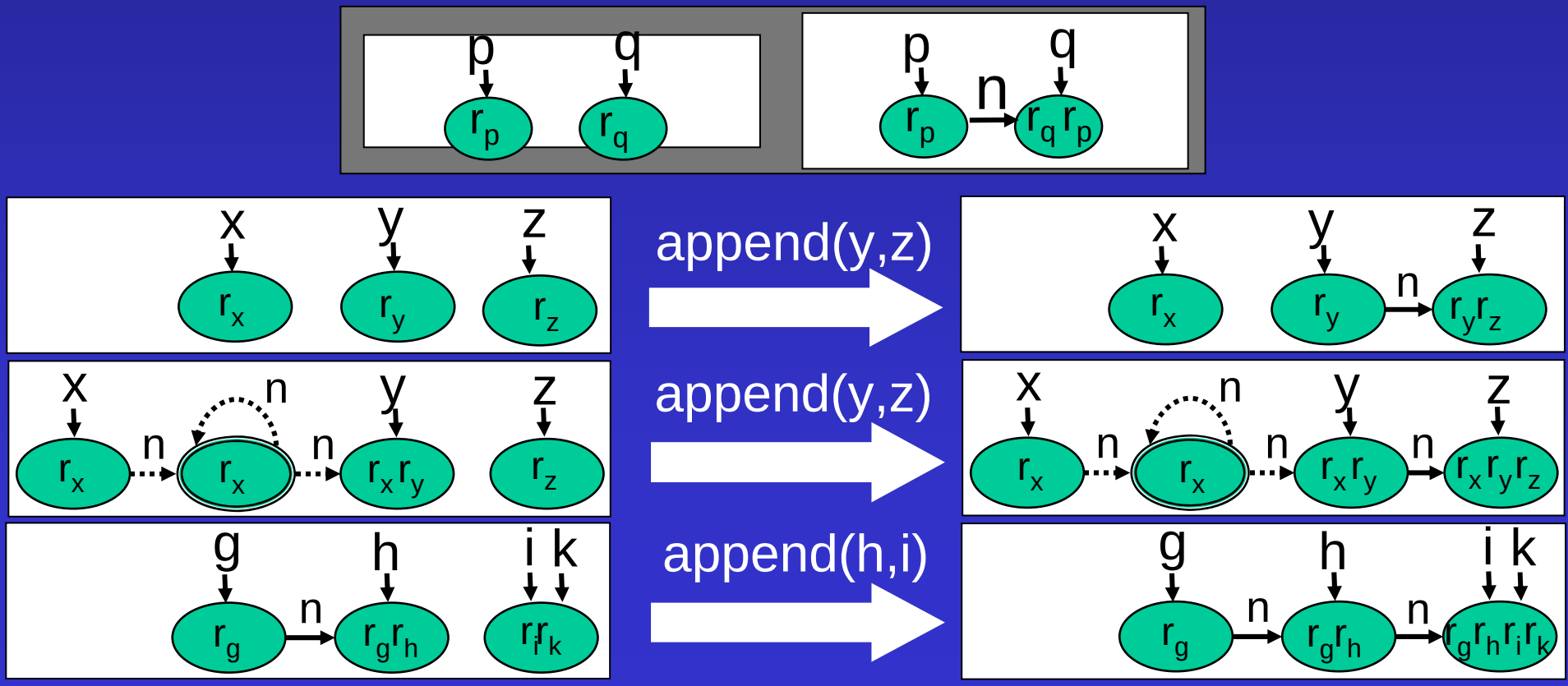
Interprocedural shape analysis

- Procedure ≡ input/output relation



Interprocedural shape analysis

- Reusable procedure summaries
  - Heap modularity



Handling Larger Programs

- Staged analysis
- Specialized abstractions
  - Counterexample guided refinement
- Coercer abstractions
  - Weaker summary nodes [Arnold, SAS'06]
  - Special join operator [Manevich, SAS'04, TACAS'07, Yang'08]
  - Heterogeneous abstractions [Yahav, PLDI'04]
- Implementation techniques
  - Optimizing transformers [Bogodlov, CAV'07]
  - Optimizing GC
  - Reducing static size
  - Partial evaluation
  - Persistent data structures [Manevich, SAS'04]
  - …