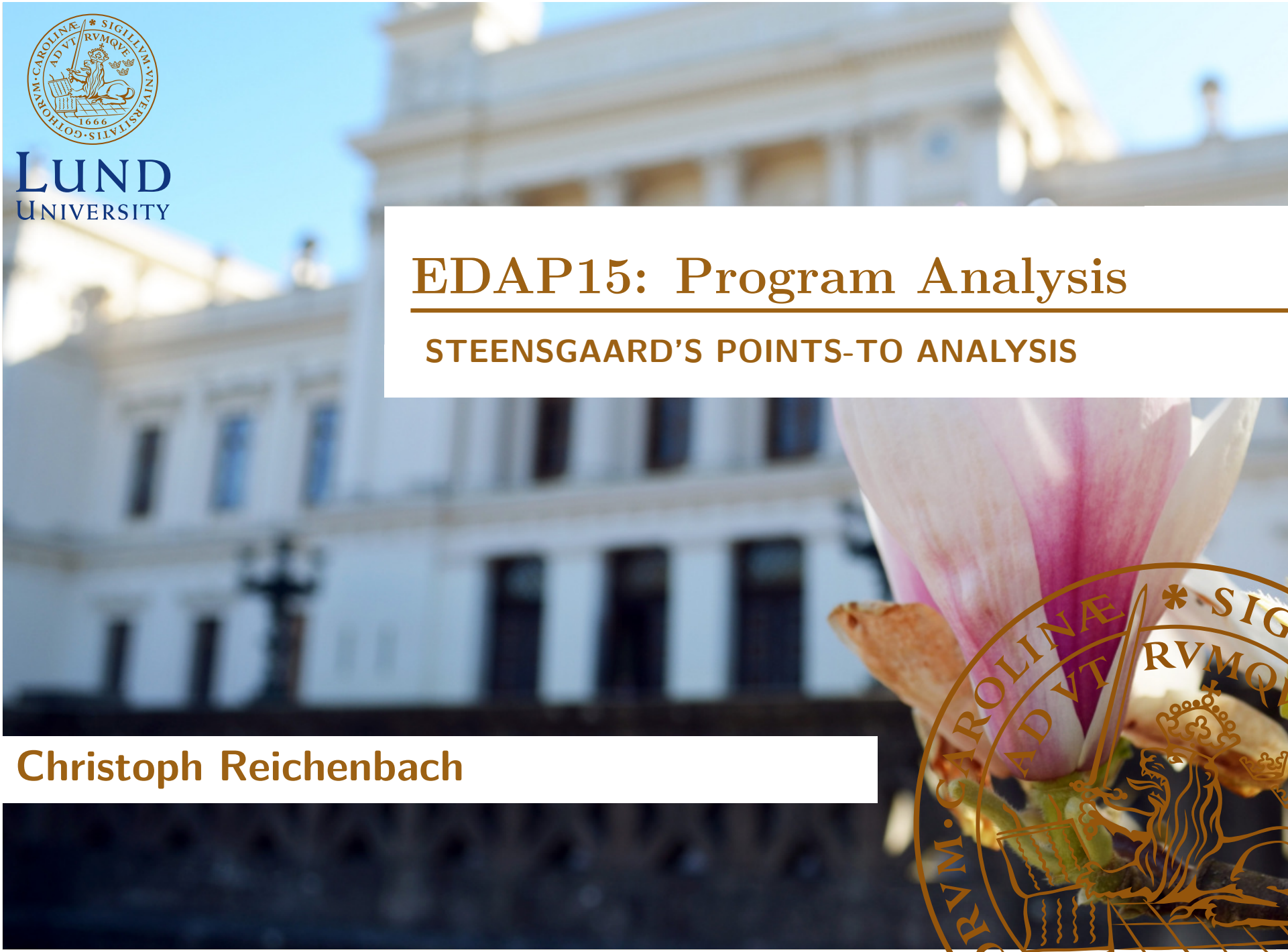


LUND UNIVERSITY

EDAP15: Program Analysis

STEENSGAARD'S POINTS-TO ANALYSIS

Christoph Reichenbach

Pointer Operations

Referencing

$\texttt{a} \rightarrow \ell$

Create, point to location:

Teal-2

```
a := new A();
a := [...];
```

Dereferencing

$\texttt{a} \rightarrow \ell \xrightarrow{f} \ell'$

Access location:

Teal-2

```
    - read -
...  := a.f;
...  := a[i];
    - write -
a.f := ...;
a[i] := ...;
```

Aliasing

$\texttt{b} \rightarrow \ell \iff \texttt{a} \rightarrow \ell$

Copy pointer:

Teal-2

```
a := b;
```

Pointer Operations Across Languages

	C	Java	Teal
Referencing	`a = &b`	`a = new A()`	`a := new A()` `a := [..., b, ...]`
Aliasing	`a = b`	`a = b`	`a := b`
Dereferencing read	`a = *b`	`a = b.f` `...`	`a := b.f` `b := a[i]`
Dereferencing write	`*a = b`	`a.f = b` `...`	`a.f := b` `a[i] := b`

Summary

- Points-to anaysis: compute Abstract Heap Graph by *approximating*

$$v \rightarrow \ell$$

- Analysis must consider:
  - **Referencing**: taking (fresh) location
    - In languages like C/C++, code can also reference locations of stack/global variables
  - **Dereferencing**: accessing object at location
  - **Aliasing**: copying location
- Locations ℓ may model different parts of memory:
  - Static variables: uniquely defined
  - Stack-dynamic variables: zero or more copies (recursion!)
  - Heap-dynamic variables: zero or more copies without variable names attached

Steensgaard's Points-To Analysis

- Fast: $O(n\alpha(n,n))$ over variables in program
  - Sacrifices Precision for speed
  - Developed to deal with large code bases at AT&T
- *Equality-based*
- Intuition:
  Whenever two variables *might* point to the same memory location, treat them as globally equal

B. Steensgaard. 'Points-to analysis in almost linear time.' In Proceecings of POPL '96, pages 32–41. ACM Press, 1996.

Distinguishing Field Names?

- For simplicity, don't distinguish field names:
- `a.`$\square$ instead of `a.f` or `a.g`
- Will discuss consequences of this simplification shortly

Constraint Collection

- 'Points-to-set': $pts(v)$ approximates $\{\ell | v \twoheadrightarrow \ell\}$
  - $pts(v) = \{\ell | v \rightarrowtriangle \ell\}$
- For each statement in program:
  - If **Referencing** (`a := new`$_{\ell_b}$ `...`) (allocation site ℓ_b)):

$$\ell_b \in pts(a)$$

  - If **Aliasing** (`a := b`):

$$pts(a) = pts(b)$$

  - If **Dereferencing read** (`a := b.`$\square$):

$$\text{for each } \ell \in pts(b) \implies pts(a) = pts(\ell)$$

  - If **Dereferencing write** (`a.`$\square$ `:= b`):

$$\text{for each } \ell \in pts(a) \implies pts(b) = pts(\ell)$$

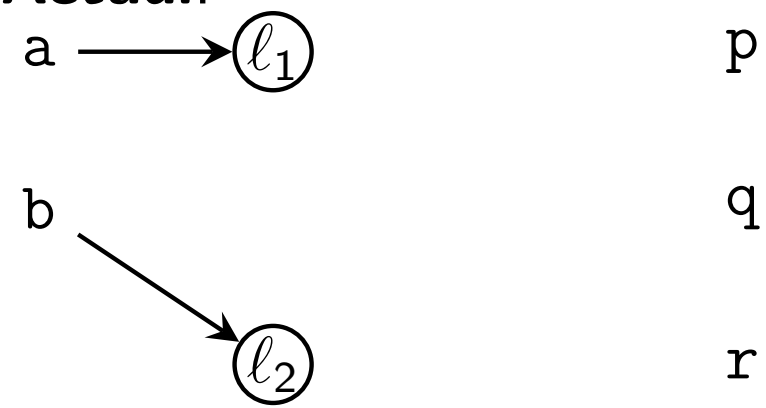
Example

⇒	`x := new`$_{\ell_z}$	$\ell_z \in pts(\mathtt{x})$
	`x := y`	$pts(\mathtt{x}) = pts(\mathtt{y})$
	`x := y.`$\square$	for each $\ell \in pts(\mathtt{y})$ $\implies$ $pts(\mathtt{x}) = pts(\ell)$
	`x.`$\square$ `:= y`	for each $\ell \in pts(\mathtt{x})$ $\implies$ $pts(\mathtt{y}) = pts(\ell)$

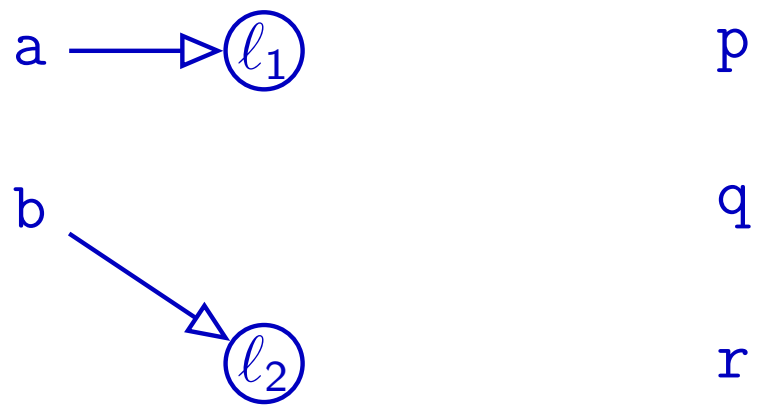
Teal

```
var a := new_ℓ1();
var b := new_ℓ2(); //⇐
a := new_ℓ3();
var p := new_ℓ4();
p.n := a;
var q := new_ℓ6();
q.n := b;
p := q;
var r := q.n;
```

- **Actual**:



- **Steensgaard**:



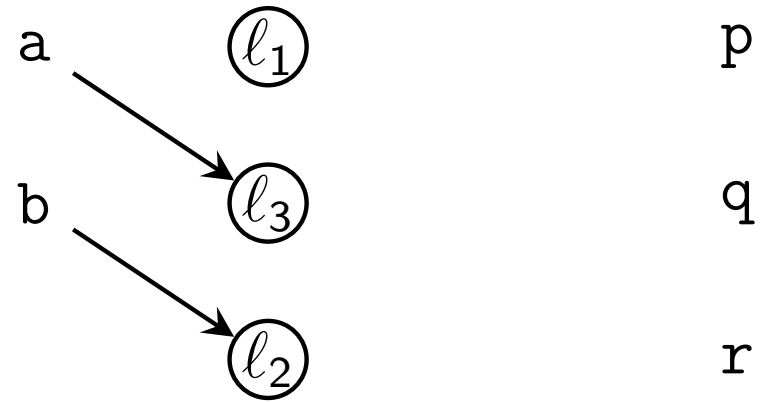
Example

⇒	`x := new`$_{\ell_z}$	$\ell_z \in pts(\mathtt{x})$
	`x := y`	$pts(\mathtt{x}) = pts(\mathtt{y})$
	`x := y.`$\square$	for each $\ell \in pts(\mathtt{y})$ $\implies$ $pts(\mathtt{x}) = pts(\ell)$
	`x.`$\square$ `:= y`	for each $\ell \in pts(\mathtt{x})$ $\implies$ $pts(\mathtt{y}) = pts(\ell)$

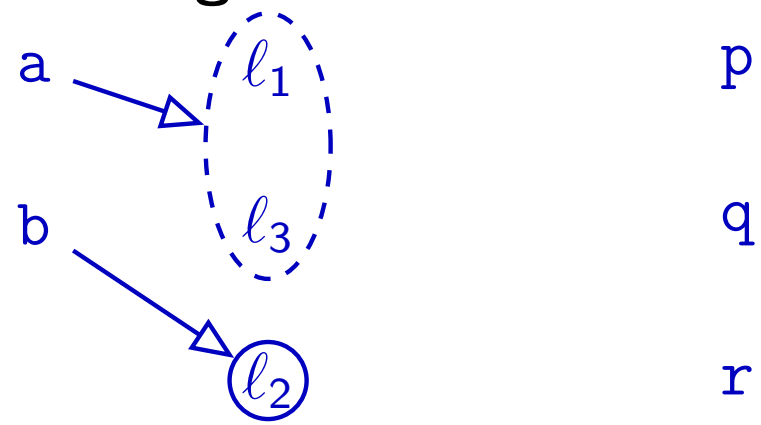
Teal

```
var a := new_ℓ1();
var b := new_ℓ2();
a := new_ℓ3();          //⇐
var p := new_ℓ4();
p.n := a;
var q := new_ℓ6();
q.n := b;
p := q;
var r := q.n;
```

- **Actual**:



- **Steensgaard**:



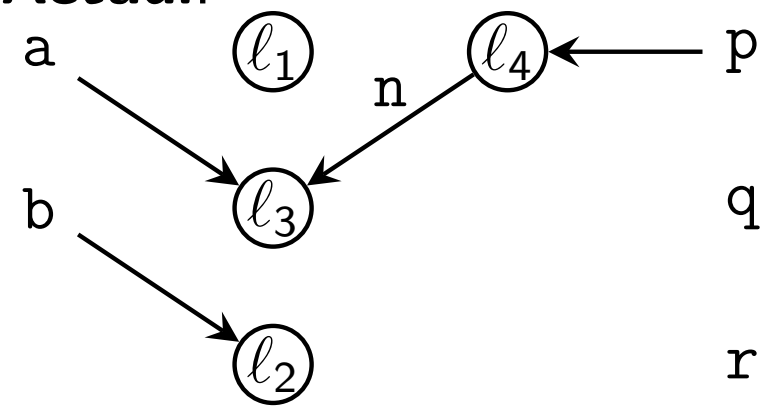
Example

	`x := new`$_{\ell_z}$	$\ell_z \in pts(\mathtt{x})$
	`x := y`	$pts(\mathtt{x}) = pts(\mathtt{y})$
	`x := y.`$\square$	for each $\ell \in pts(\mathtt{y})$ $\implies pts(\mathtt{x}) = pts(\ell)$
$\Rightarrow$	`x.`$\square$ `:= y`	for each $\ell \in pts(\mathtt{x})$ $\implies pts(\mathtt{y}) = pts(\ell)$

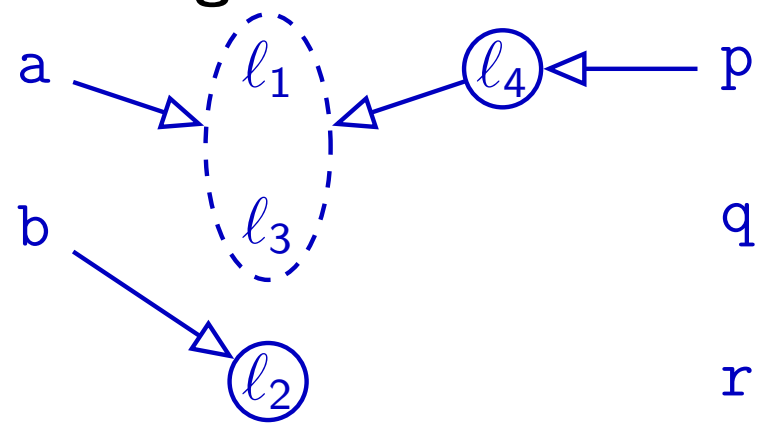
Teal

```
var a := new_ℓ1();
var b := new_ℓ2();
a := new_ℓ3();
var p := new_ℓ4();
p.n := a;                    //⇐
var q := new_ℓ6();
q.n := b;
p := q;
var r := q.n;
```

- **Actual**:



- **Steensgaard**:



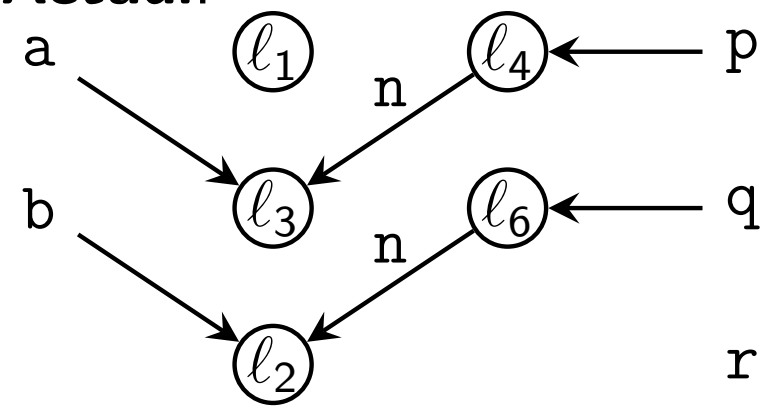
Example

	`x := new`$_{\ell_z}$	$\ell_z \in pts(\mathtt{x})$
	`x := y`	$pts(\mathtt{x}) = pts(\mathtt{y})$
	`x := y.`$\square$	for each $\ell \in pts(\mathtt{y})$ $\implies pts(\mathtt{x}) = pts(\ell)$
$\Rightarrow$	`x.`$\square$ `:= y`	for each $\ell \in pts(\mathtt{x})$ $\implies pts(\mathtt{y}) = pts(\ell)$

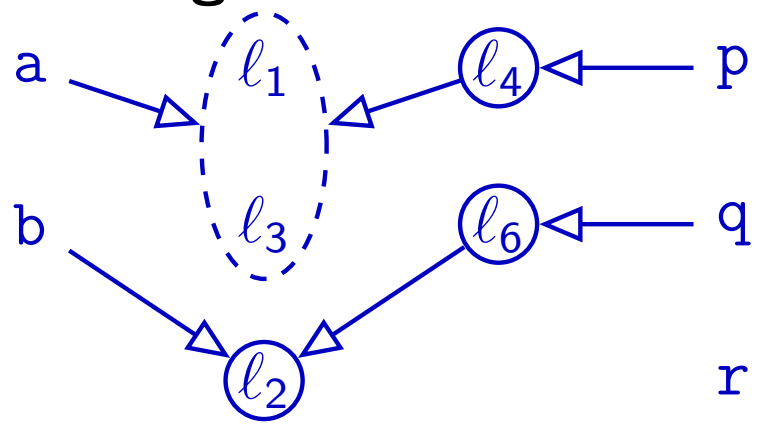
Teal

```
var a := new_ℓ1();
var b := new_ℓ2();
a := new_ℓ3();
var p := new_ℓ4();
p.n := a;
var q := new_ℓ6();
q.n := b;                    //⇐
p := q;
var r := q.n;
```

- **Actual**:



- **Steensgaard**:



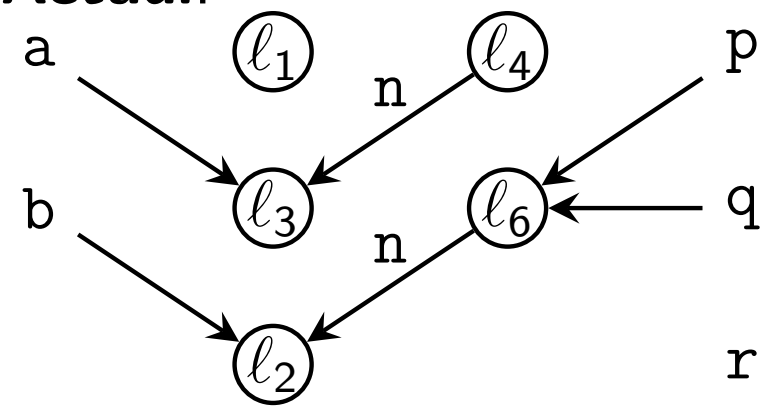
Example

	`x := new`$_{\ell_z}$	$\ell_z \in pts(\mathtt{x})$
$\Rightarrow$	`x := y`	$pts(\mathtt{x}) = pts(\mathtt{y})$
	`x := y.`$\square$	for each $\ell \in pts(\mathtt{y})$ $\implies$ $pts(\mathtt{x}) = pts(\ell)$
	`x.`$\square$ `:= y`	for each $\ell \in pts(\mathtt{x})$ $\implies$ $pts(\mathtt{y}) = pts(\ell)$

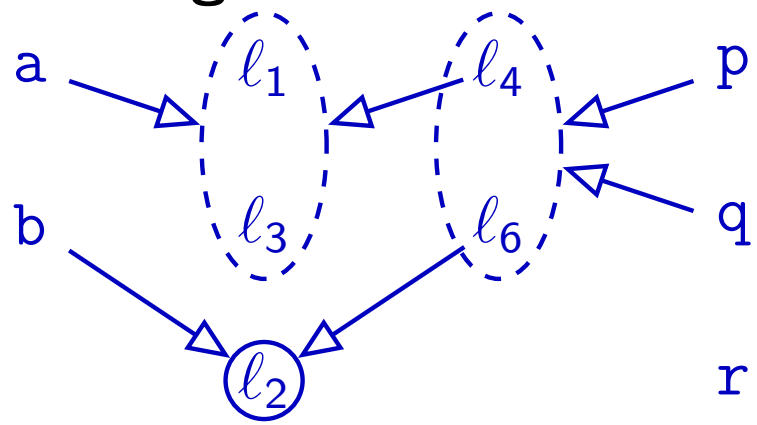
Teal

```
var a := new_ℓ1();
var b := new_ℓ2();
a := new_ℓ3();
var p := new_ℓ4();
p.n := a;
var q := new_ℓ6();
q.n := b;
p := q;                    //⇐
var r := q.n;
```

- **Actual**:



- **Steensgaard**:



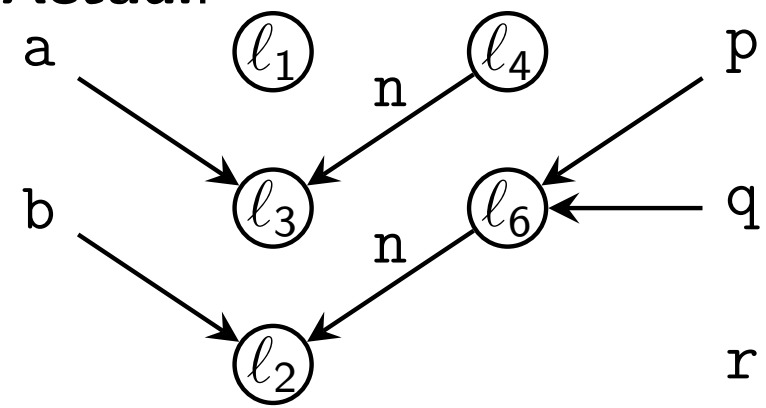
Example

	`x := new`$_{\ell_z}$	$\ell_z \in pts(\mathtt{x})$
$\Rightarrow$	`x := y`	$pts(\mathtt{x}) = pts(\mathtt{y})$
	`x := y.`$\square$	for each $\ell \in pts(\mathtt{y})$ $\implies$ $pts(\mathtt{x}) = pts(\ell)$
	`x.`$\square$ `:= y`	for each $\ell \in pts(\mathtt{x})$ $\implies$ $pts(\mathtt{y}) = pts(\ell)$

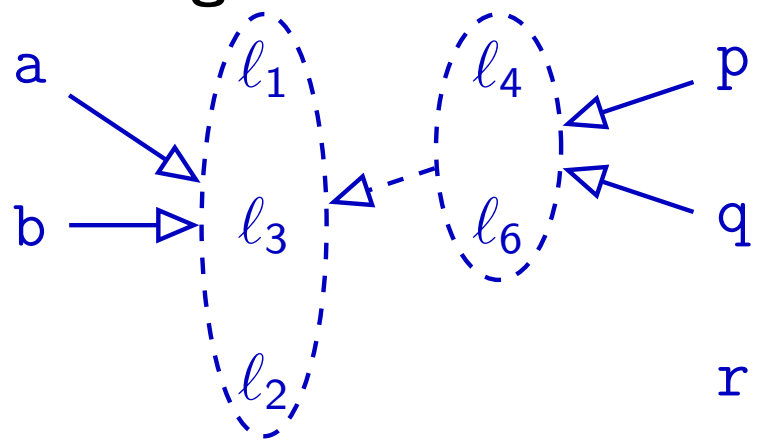
Teal

```
var a := new_ℓ1();
var b := new_ℓ2();
a := new_ℓ3();
var p := new_ℓ4();
p.n := a;
var q := new_ℓ6();
q.n := b;
p := q;                    //⇐
var r := q.n;
```

- **Actual**:



- **Steensgaard**:



When merging: 'collapse' children (merge recursively)

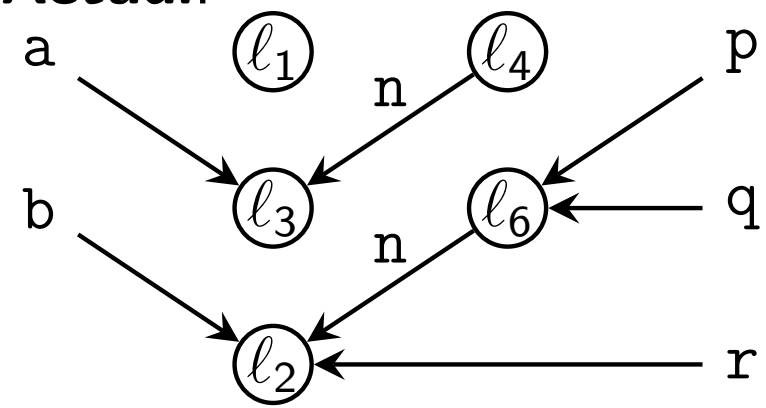
Example

	`x := new`$_{\ell_z}$	$\ell_z \in pts(\mathtt{x})$
	`x := y`	$pts(\mathtt{x}) = pts(\mathtt{y})$
$\Rightarrow$	`x := y.`$\square$	for each $\ell \in pts(\mathtt{y})$ $\implies pts(\mathtt{x}) = pts(\ell)$
	`x.`$\square$ `:= y`	for each $\ell \in pts(\mathtt{x})$ $\implies pts(\mathtt{y}) = pts(\ell)$

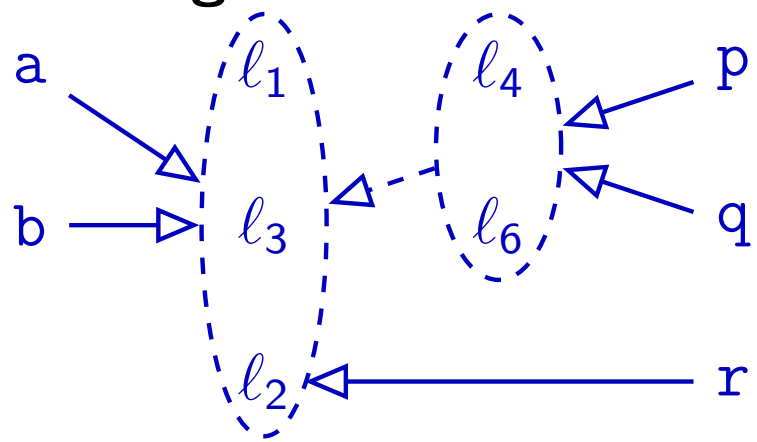
Teal

```
var a := new_ℓ1();
var b := new_ℓ2();
a := new_ℓ3();
var p := new_ℓ4();
p.n := a;
var q := new_ℓ6();
q.n := b;
p := q;
var r := q.n;      //⇐
```

- **Actual**:



- **Steensgaard**:



When merging: 'collapse' children (merge recursively)

Summary

- Points-to sets $pts(v) = \{\ell | v \rightarrowtriangle \ell\}$
- Steensgaard's points-to analysis:
  - special case of *type analysis*
- Steensgaard's analysis in practice:
  - Highly efficient when implemented with Union-Find
  - Less precise than other commonly-used analyses