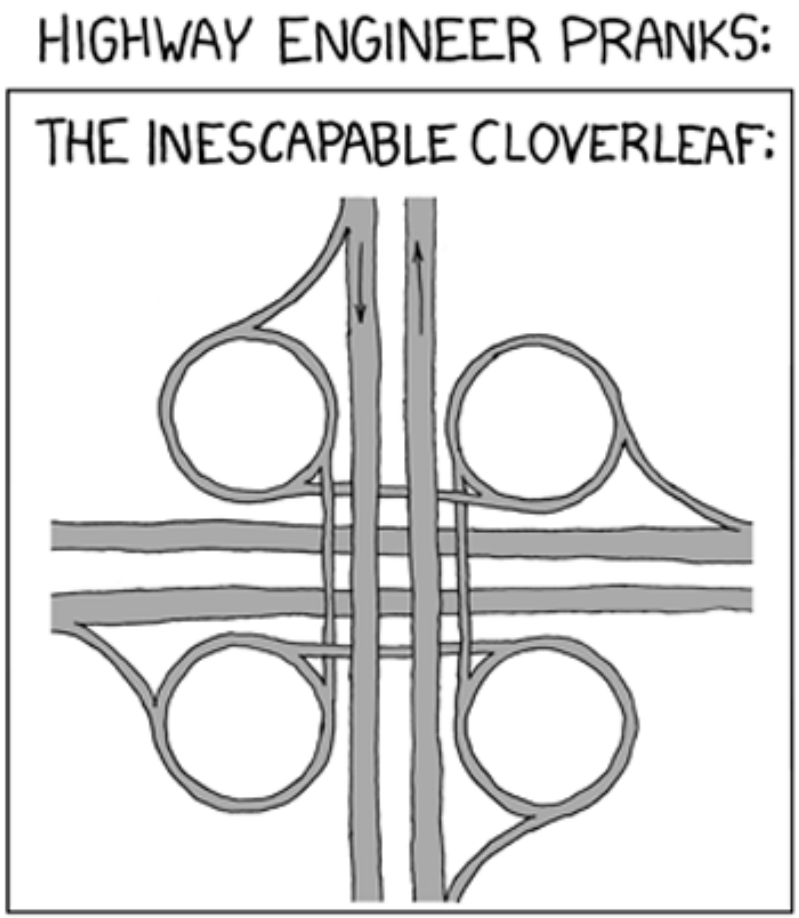
Class-Modular, Class-Escape and Points-to Analysis for Object-Oriented Languages

Alexander Herz, Kalmer Apinis Department of Computer Science Chair for Languages and Structures, Prof. Seidl

There is no (undetected) escape ...



Outline

1. Overview
2. Motivation: Encapsulation
3. Traditional Points-to Analyses
4. Analysis (Framework)
5. Results

Outline

Extract class-modular, points-to [Andersen(1994), Steensgaard(1996)] and class-escape information for a class (hierarchy)

- What does class-escape mean?
- What does class-modular mean?

Quick Reminder: Encapsulation

Example

```
struct Point{ int x,y; };
class SimpleRect{
  private:Point *ul;
  public:Point *lr;
  SimpleRect(int x1,int y1, int x2, int y2){
    ul=new Point();lr=new Point();
    ul→x=x1;ul→y=y1;lr→x=x2;lr→y=y2;}
  Point* DoEscape(){ return ul;}
};

void main(){
  SimpleRect r = new SimpleRect(0,0,10,10);
  r→lr→x = 0; //ok, lr is public
  r→ul→x = 0; //compiler error: ul is not public
  r→DoEscape()→x = 0; //no error, but private is bypassed!
}
```

Quick Reminder: Encapsulation

Example

```
struct Point{ int x,y; };
class SimpleRect{
  private:Point *ul;
  public:Point *lr;
  SimpleRect(int x1,int y1, int x2, int y2){
    ul=new Point();lr=new Point();
    ul→x=x1;ul→y=y1;lr→x=x2;lr→y=y2;}
  Point* DoEscape(){ return ul;}
};

void main(){
  SimpleRect r = new SimpleRect(0,0,10,10);
  r→lr→x = 0; //ok, lr is public
  r→ul→x = 0; //compiler error: ul is not public
  r→DoEscape()→x = 0; //no error, but private is bypassed!
}
```

Quick Reminder: Encapsulation

Example

```
struct Point{ int x,y; };
class SimpleRect{
  private:Point *ul;
  public:Point *lr;
  SimpleRect(int x1,int y1, int x2, int y2){
    ul=new Point();lr=new Point();
    ul→x=x1;ul→y=y1;lr→x=x2;lr→y=y2;}
  Point* DoEscape(){ return ul;}
};

void main(){
  SimpleRect r = new SimpleRect(0,0,10,10);
  r→lr→x = 0; //ok, lr is public
  r→ul→x = 0; //compiler error: ul is not public
  r→DoEscape()→x = 0; //no error, but private is bypassed!
}
```

Quick Reminder: Encapsulation

Example

```
struct Point{ int x,y; };
class SimpleRect{
  private:Point *ul;
  public:Point *lr;
  SimpleRect(int x1,int y1, int x2, int y2){
    ul=new Point();lr=new Point();
    ul→x=x1;ul→y=y1;lr→x=x2;lr→y=y2;}
  Point* DoEscape(){ return ul;}
};

void main(){
  SimpleRect r = new SimpleRect(0,0,10,10);
  r→lr→x = 0; //ok, lr is public
  r→ul→x = 0; //compiler error: ul is not public
  r→DoEscape()→x = 0; //no error, but private is bypassed!
}
```

Quick Reminder: Encapsulation

Example

```
struct Point{ int x,y; };
class SimpleRect{
  private:Point *ul;
  public:Point *lr;
  SimpleRect(int x1,int y1, int x2, int y2){
    ul=new Point();lr=new Point();
    ul→x=x1;ul→y=y1;lr→x=x2;lr→y=y2;}
  Point* DoEscape(){ return ul;}
};

void main(){
  SimpleRect r = new SimpleRect(0,0,10,10);
  r→lr→x = 0; //ok, lr is public
  r→ul→x = 0; //compiler error: ul is not public
  r→DoEscape()→x = 0; //no error, but private is bypassed!
}
```

Quick Reminder: Encapsulation

Example

```
struct Point{ int x,y; };
class SimpleRect{
  private:Point *ul;
  public:Point *lr;
  SimpleRect(int x1,int y1, int x2, int y2){
    ul=new Point();lr=new Point();
    ul→x=x1;ul→y=y1;lr→x=x2;lr→y=y2;}
  Point* DoEscape(){ return ul;}
};

void main(){
  SimpleRect r = new SimpleRect(0,0,10,10);
  r→lr→x = 0; //ok, lr is public
  r→ul→x = 0; //compiler error: ul is not public
  r→DoEscape()→x = 0; //no error, but private is bypassed!
}
```

Quick Reminder: Encapsulation

Example

```
struct Point{ int x,y; };
class SimpleRect{
  private:Point *ul;
  public:Point *lr;
  SimpleRect(int x1,int y1, int x2, int y2){
    ul=new Point();lr=new Point();
    ul→x=x1;ul→y=y1;lr→x=x2;lr→y=y2;}
  Point* DoEscape(){ return ul;}
};

void main(){
  SimpleRect r = new SimpleRect(0,0,10,10);
  r→lr→x = 0; //ok, lr is public
  r→ul→x = 0; //compiler error: ul is not public
  r→DoEscape()→x = 0; //no error, but private is bypassed!
}
```

Quick Reminder: Encapsulation

Example

```
struct Point{ int x,y; };
class SimpleRect{
  private:Point *ul;
  public:Point *lr;
  SimpleRect(int x1,int y1, int x2, int y2){
    ul=new Point();lr=new Point();
    ul→x=x1;ul→y=y1;lr→x=x2;lr→y=y2;}
  Point* DoEscape(){ return ul;}
};

void main(){
  SimpleRect r = new SimpleRect(0,0,10,10);
  r→lr→x = 0; //ok, lr is public
  r→ul→x = 0; //compiler error: ul is not public
  r→DoEscape()→x = 0; //no error, but private is bypassed!
}
```

Quick Reminder: Encapsulation

Example

```
struct Point{ int x,y; };
class SimpleRect{
  private:Point *ul;
  public:Point *lr;
  SimpleRect(int x1,int y1, int x2, int y2){
    ul=new Point();lr=new Point();
    ul→x=x1;ul→y=y1;lr→x=x2;lr→y=y2;}
  Point* DoEscape(){ return ul;}
};

void main(){
  SimpleRect r = new SimpleRect(0,0,10,10);
  r→lr→x = 0; //ok, lr is public
  r→ul→x = 0; //compiler error: ul is not public
  r→DoEscape()→x = 0; //no error, but private is bypassed!
}
```

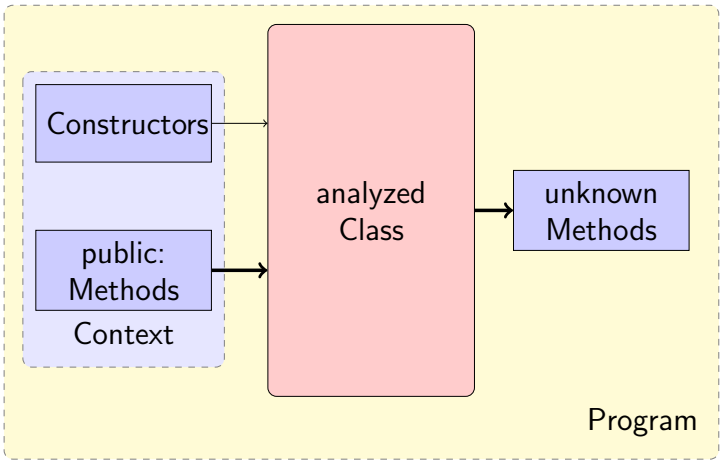
Quick Reminder: Encapsulation

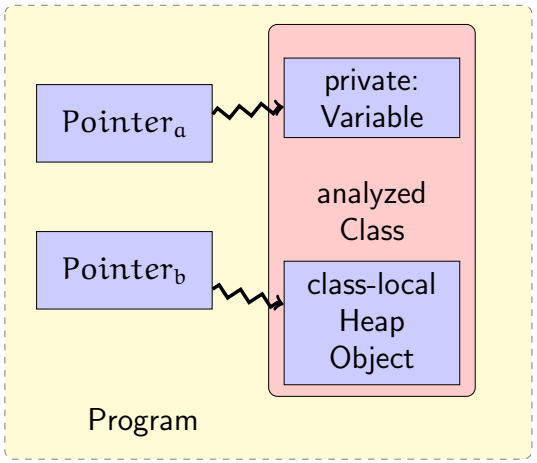
Example

```
struct Point{ int x,y; };
class SimpleRect{
  private:Point *ul;
  public:Point *lr;
  SimpleRect(int x1,int y1, int x2, int y2){
    ul=new Point();lr=new Point();
    ul→x=x1;ul→y=y1;lr→x=x2;lr→y=y2;}
  Point* DoEscape(){ return ul;}
};

void main(){
  SimpleRect r = new SimpleRect(0,0,10,10);
  r→lr→x = 0; //ok, lr is public
  r→ul→x = 0; //compiler error: ul is not public
  r→DoEscape()→x = 0; //no error, but private is bypassed!
}
```

Class-Modular/Class-Escape





- Class-Modular : the context that uses the class is not available (e.g. main())
- Class-Escape : private variables and local heap objects that become accessible from outside the class (e.g. ul); compare [Blanchet(1999)]

Outline

Encapsulation is important for Industry and Science

- Industry
  - C++
  - Java

Encapsulation is important for Industry and Science

- Industry
  - C++
  - Java
- Scientific Tools
  - Gromacs
  - Libmesh
  - Numerical Recipes
  - ...

Encapsulation is important for Industry and Science

- Industry
  - C++
  - Java
- Scientific Tools
  - Gromacs
  - Libmesh
  - Numerical Recipes
  - ...
- Encapsulation commonly enforced [Sutter and Alexandrescu(2005)]

Outline

Problems with whole-program or function-modular Points-to Analysis

- Only invoked methods are analyzed (no class-invariant)

Problems with whole-program or function-modular Points-to Analysis

- Only invoked methods are analyzed (no class-invariant)
- Code may be analyzed unnecessarily (context, resolve vtable)

Problems with whole-program or function-modular Points-to Analysis

- Only invoked methods are analyzed (no class-invariant)
- Code may be analyzed unnecessarily (context, resolve vtable)
- Analyzing classes without context is not easily possible with acceptable precision

Outline

Framework

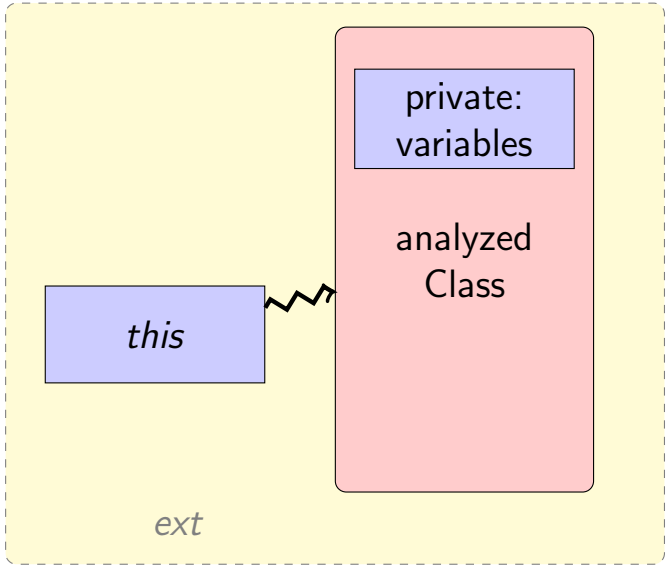
- Framework transforms a (traditional) points-to analysis into class-modular, class-escape and points-to analysis
- Plug-in given as abstract semantic function and ability to querry points-to sets (virtually all existing sound analyses)
- Plug-in internals unknown (e.g. domain type) ... see paper
- Can pick plug-in with most favorable properties

Analysis

- Analyze one class(-hierarchy) at a time
- Add two special memory locations to the analysis domain (not part of the class)
- *ext* abstracts all memory addresses existing outside of the class
- All addresses reachable from *ext* have escaped
- *this* abstracts all possible class instances
- *ext* and fields of *this* are treated flow insensitively

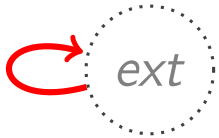
Partitioned Memory

```
class Rect
{
private:  Point *ul,*lr;
          Point **e,**l;
          Point ***p;
          Point *priv;
public:   Point *pub;
//...
Rect(int x1,int y1, int x2, int y2)
{
  ul=new Point();//Pt_a
  lr=new Point();//Pt_b
  p=&e;e=&lr;l=&ul;
  ul→x=x1;ul→y=y1;
  lr→x=x2;lr→y=y2;
}
//...
```



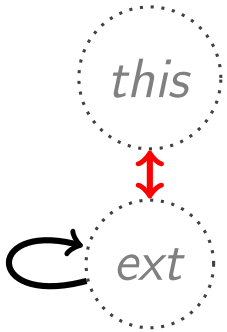
Init Domain

```
class Rect
{
private:  Point *ul,*lr;
          Point **e,**l;
          Point ***p;
          Point *priv;
public:   Point *pub;
//...
Rect(int x1,int y1, int x2, int y2)
{
  ul=new Point();//Pt_a
  lr=new Point();//Pt_b
  p=&e;e=&lr;l=&ul;
  ul→x=x1;ul→y=y1;
  lr→x=x2;lr→y=y2;
}
//...
```



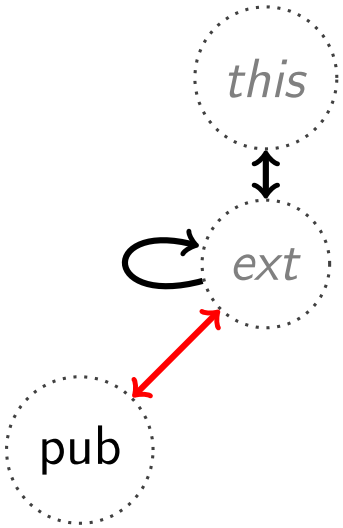
New (not seen by analysis)

```
class Rect
{
private:  Point *ul,*lr;
          Point **e,**l;
          Point ***p;
          Point *priv;
public:   Point *pub;
//...
Rect(int x1,int y1, int x2, int y2)
{
  ul=new Point();//Pt_a
  lr=new Point();//Pt_b
  p=&e;e=&lr;l=&ul;
  ul→x=x1;ul→y=y1;
  lr→x=x2;lr→y=y2;
}
//...
```



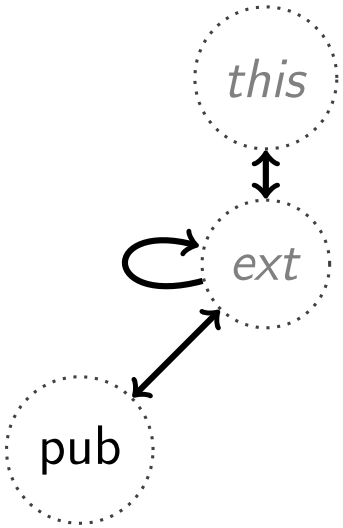
Public Fields

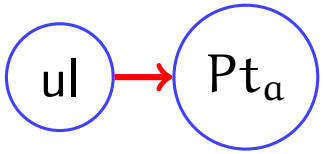
```
class Rect
{
private:   Point *ul,*lr;
           Point **e,**l;
           Point ***p;
           Point *priv;
public:    Point *pub;
//...
Rect(int x1,int y1, int x2, int y2)
{
  ul=new Point();//Pt_a
  lr=new Point();//Pt_b
  p=&e;e=&lr;l=&ul;
  ul→x=x1;ul→y=y1;
  lr→x=x2;lr→y=y2;
}
//...
```



Constructor(s)

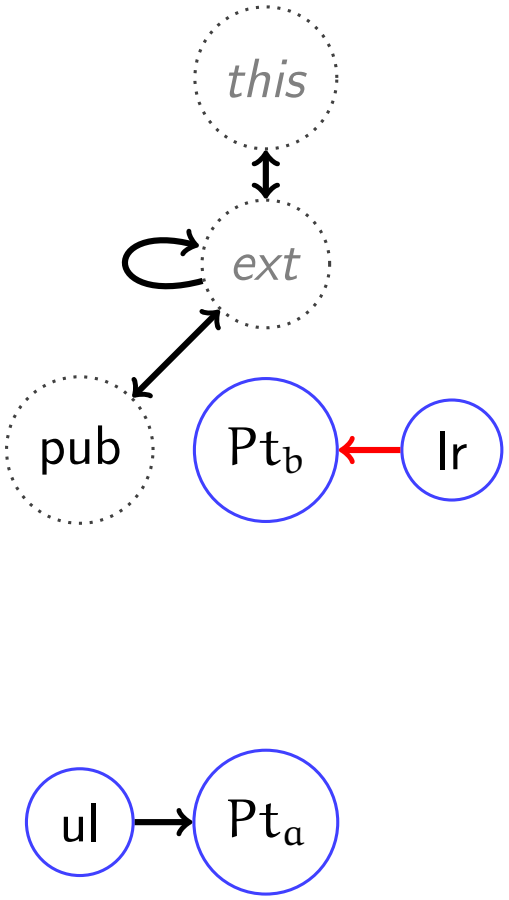
```
class Rect
{
private:  Point *ul,*lr;
          Point **e,**l;
          Point ***p;
          Point *priv;
public:   Point *pub;
//...
Rect(int x1,int y1, int x2, int y2)
{
  ul=new Point();//Pt_a
  lr=new Point();//Pt_b
  p=&e;e=&lr;l=&ul;
  ul→x=x1;ul→y=y1;
  lr→x=x2;lr→y=y2;
}
//...
```





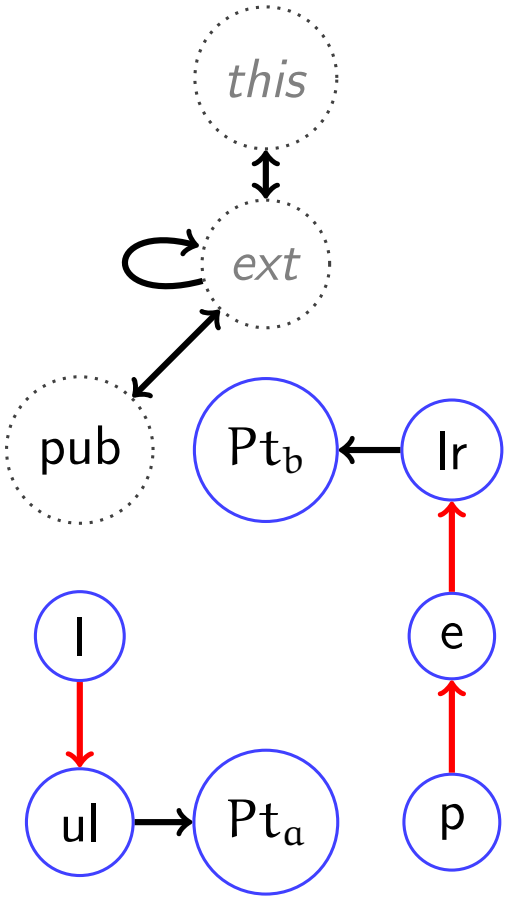
Constructor(s)

```
class Rect
{
private:  Point *ul,*lr;
          Point **e,**l;
          Point ***p;
          Point *priv;
public:   Point *pub;
//...
Rect(int x1,int y1, int x2, int y2)
{
  ul=new Point();//Pt_a
  lr=new Point();//Pt_b
  p=&e;e=&lr;l=&ul;
  ul→x=x1;ul→y=y1;
  lr→x=x2;lr→y=y2;
}
//...
```



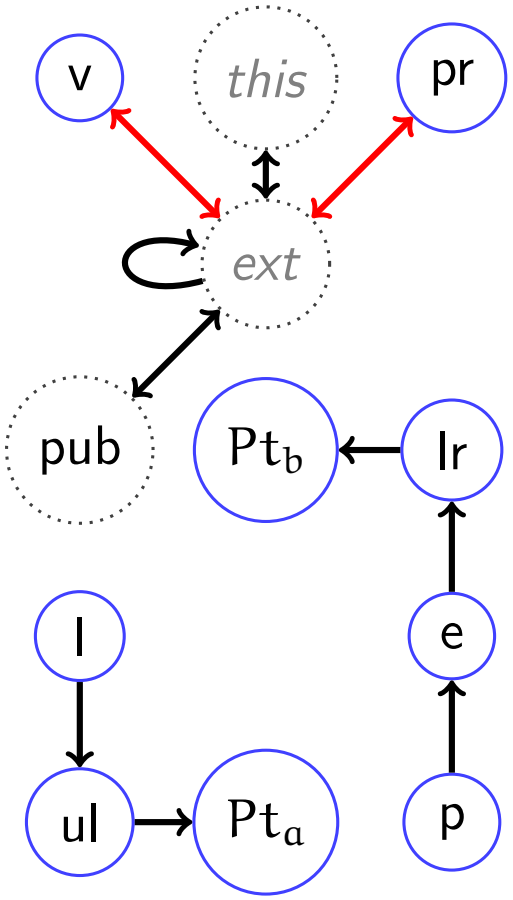
Constructor(s)

```
class Rect
{
private:   Point *ul,*lr;
           Point **e,**l;
           Point ***p;
           Point *priv;
public:    Point *pub;
//...
Rect(int x1,int y1, int x2, int y2)
{
  ul=new Point();//Pt_a
  lr=new Point();//Pt_b
  p=&e;e=&lr;l=&ul;
  ul→x=x1;ul→y=y1;
  lr→x=x2;lr→y=y2;
}
//...
```



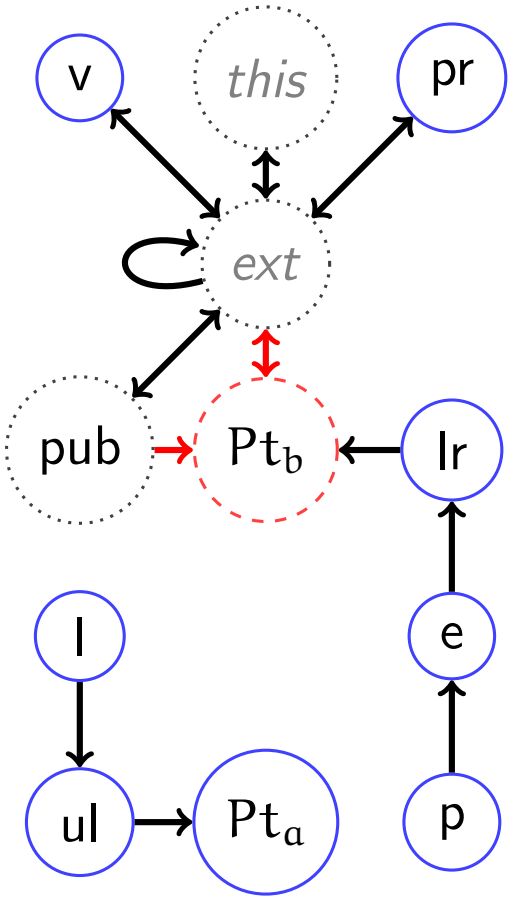
Public methods $[\![ext = \mathrm{DoEscape}(ext, ext)]\!]^\sharp$

```
//...
Point** DoEscape(Point**v,Rect* pr)
{ //pt_b escapes in the following:
  pub=lr;
  pr→priv=lr;
  unknown(lr);
  *v=*e;
  return e;
}
};
```



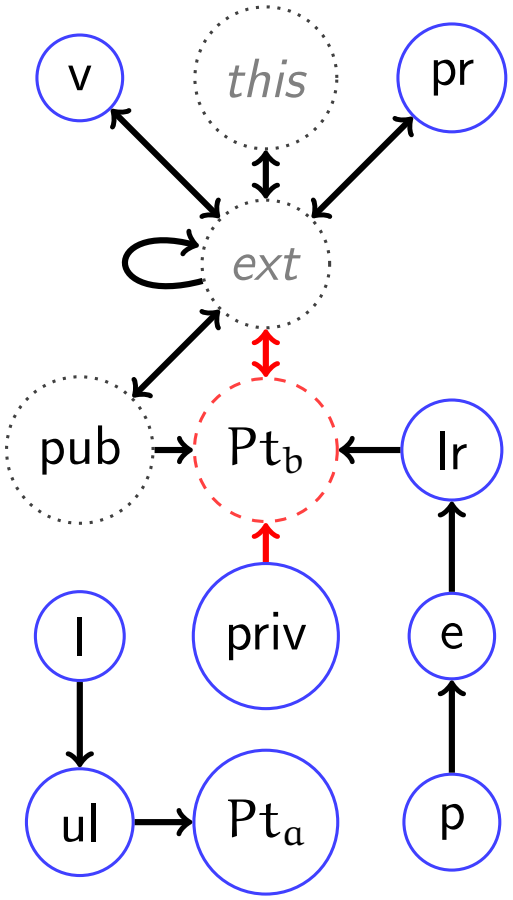
Assignment to externally accessible variable

```
//...
Point** DoEscape(Point**v,Rect* pr)
{ //pt_b escapes in the following:
  pub=lr;
  pr→priv=lr;
  unknown(lr);
  *v=*e;
  return e;
}
};
```



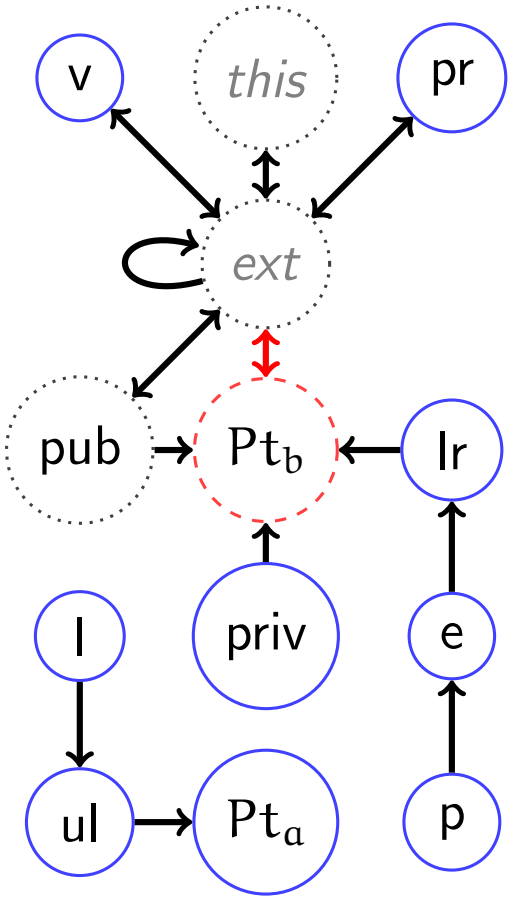
pr == this or other instance?

```
//...
Point** DoEscape(Point**v,Rect* pr)
{ //pt_b escapes in the following:
  pub=lr;
  pr→priv=lr;
  unknown(lr);
  *v=*e;
  return e;
}
};
```



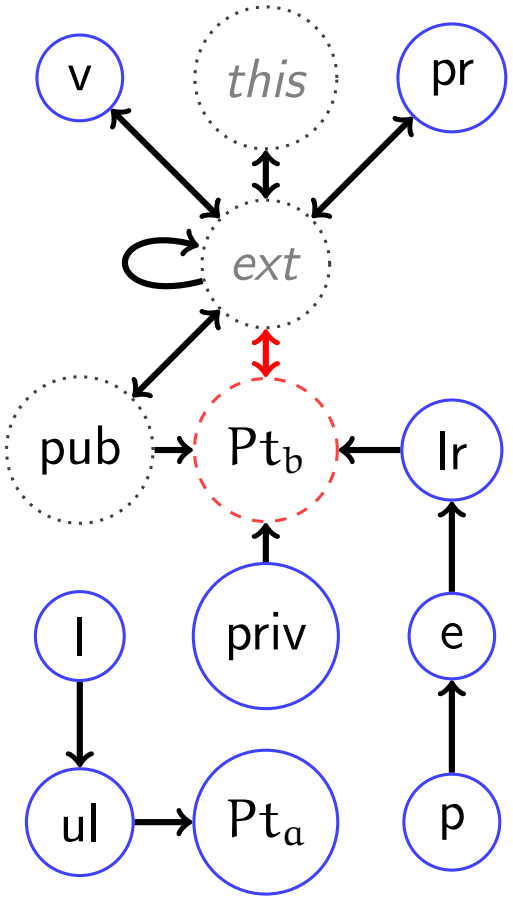
Over-approximate unknown code!

```
//...
Point** DoEscape(Point**v,Rect* pr)
{ //pt_b escapes in the following:
  pub=lr;
  pr→priv=lr;
  unknown(lr);
  *v=*e;
  return e;
}
};
```



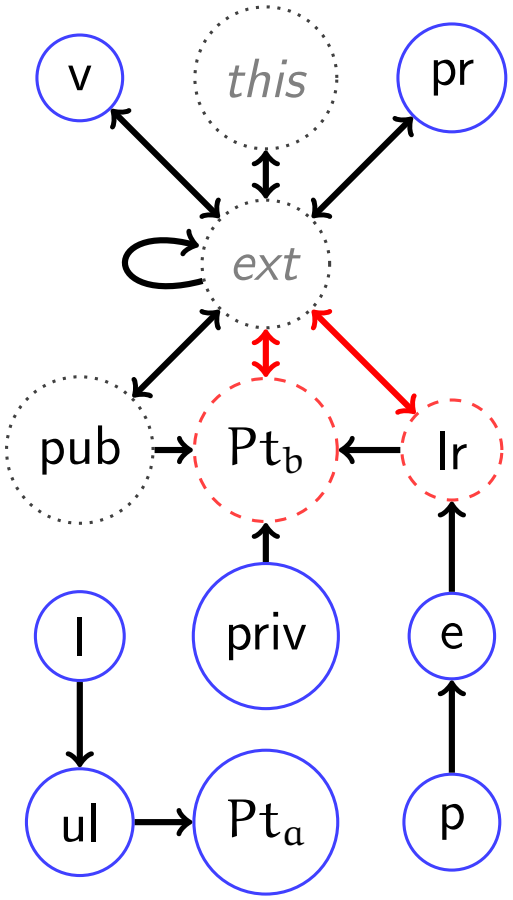
Assign to method argument

```
//...
Point** DoEscape(Point**v,Rect* pr)
{ //pt_b escapes in the following:
  pub=lr;
  pr→priv=lr;
  unknown(lr);
  *v=*e;
  return e;
}
};
```



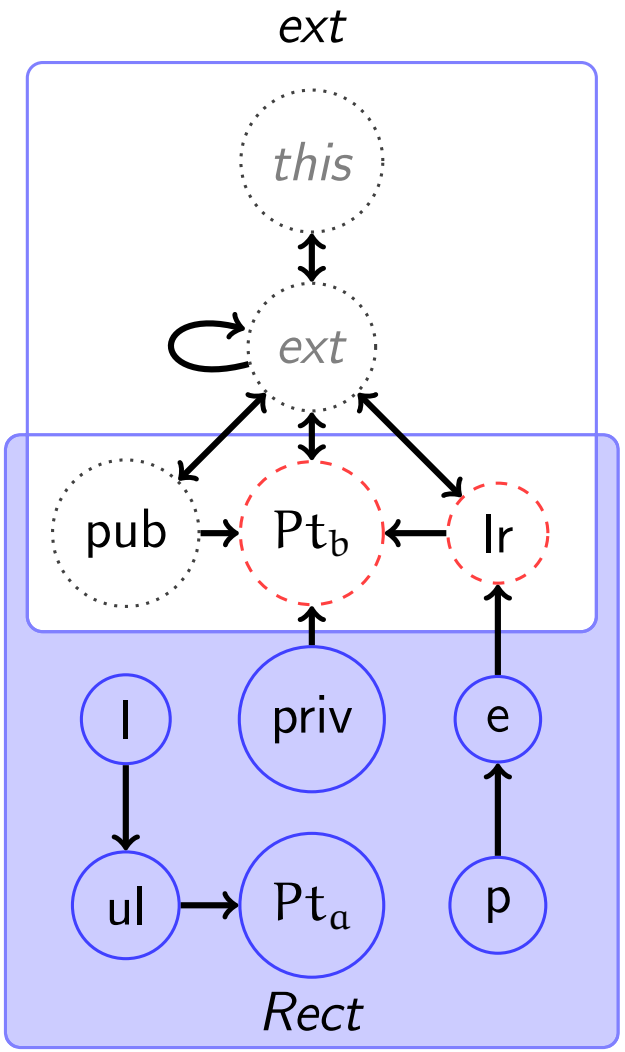
Assign to ext

```
//...
Point** DoEscape(Point**v,Rect* pr)
{ //pt_b escapes in the following:
  pub=lr;
  pr→priv=lr;
  unknown(lr);
  *v=*e;
  return e;
}
};
```



Finally ...

```
//...
Point** DoEscape(Point**v,Rect* pr)
{ //pt_b escapes in the following:
  pub=lr;
  pr→priv=lr;
  unknown(lr);
  *v=*e;
  return e;
}
};
```



Global Fix-Point

- (Re-)analyze all public methods until globals do not change anymore
- Over-approximates calling all public methods in any order with all possible input data
- Solution is a class-invariant which holds for any context the class may be used in

Outline

Results

Code	Classes	C++[loc]	C[loc]	Time[s]	Ext[%]
Industrial	44	28.566	1.368.112	282	23
Ogre	134	71.886	1.998.910	42	25

- Unmodified real world C++ code (incomplete and hard to get)
- C++ code is lowered to (semantically equivalent) C code for Goblint via LLVM
- Ogre (open source 3d engine) code is less complex (C-loc/C++-loc)
- Time: summed for all classes
- Ext: average of private fields that may escape over all classes
- Verified warnings for Nokia Siemens Networks (Industrial Code)
- Guided decision to apply software transformation at NSN

Related Work

- Object In-lining [Dolby and Chien(2000)]
- Class Invariants [Logozzo(2003)]
- Class-modular compiler optimizations
- Static analysis of C++ libraries

Thank You

[Andersen(1994)] L. Andersen.
Program analysis and specialization for the C programming language.
Technical report, 94-19, University of Copenhagen, 1994.

[Blanchet(1999)] B. Blanchet.
Escape analysis for object-oriented languages: application to Java.
SIGPLAN Not., 34:20–34, 1999.
ISSN 0362-1340.

[Dolby and Chien(2000)] J. Dolby and A. Chien.
An automatic object inlining optimization and its evaluation.
SIGPLAN Not., 35(5):345–357, 2000.

[Logozzo(2003)] F. Logozzo.
Class-Level Modular Analysis for Object Oriented Languages.
In R. Cousot, editor, *Static Analysis*, volume 2694, pages 37–54. Springer Berlin / Heidelberg, 2003.
ISBN 978-3-540-40325-8.

[Steensgaard(1996)] B. Steensgaard.
Points-to analysis in almost linear time.
In *Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '96, pages 32–41, New York, NY, USA, 1996. ACM.
ISBN 0-89791-769-3.

[Sutter and Alexandrescu(2005)] H. Sutter and A. Alexandrescu.
C++ coding standards: 101 rules, guidelines, and best practices.
Addison-Wesley Professional, 2005.