Semi-Automatic Region-Based Memory Management for Real-Time Java Embedded Systems

G. Salagnac, C. Rippert, S. Yovine

Verimag Lab.
Université Joseph Fourier
Grenoble, France

http://www-verimag.imag.fr/~salagnac

salagnac@imag.fr

Motivation

The Java programming language

- Attractive language
  - Automatic memory management

Implementation pitfalls

- Garbage Collection $\Rightarrow$ pause times and fragmentation

$\Longrightarrow$ Difficult to use in a real-time embedded context

Our approach

Non-determinism of Garbage Collector pause times:
the problem is in the JVM, not in the language!

Proposition:

- Keep the language
  - no *manual* memory management
- Change the implementation
  - replace the GC by a *predictable* allocator
    - use region-based memory management
  - automatically compute object lifetimes at compilation
    - undecidable problem
    - find a reasonable over-approximation

Our approach

Non-determinism of Garbage Collector pause times:
the problem is in the JVM, not in the language!

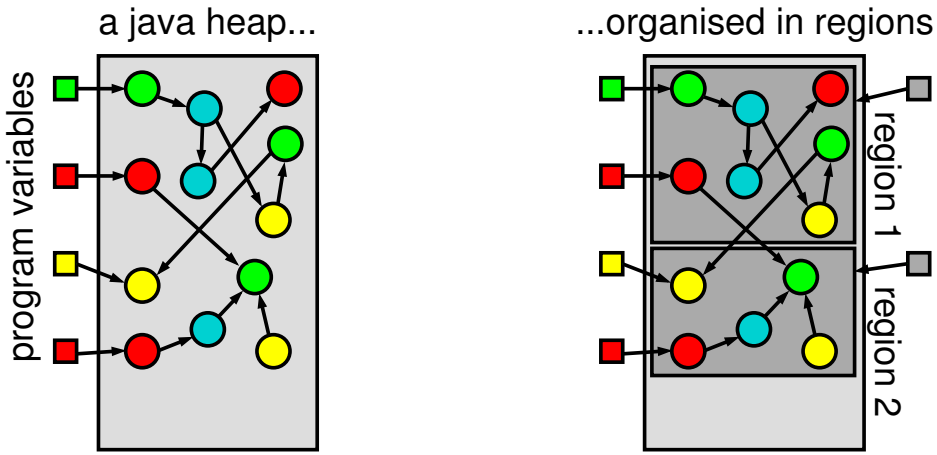
Proposition:

- Keep the language
  - no *manual* memory management
- Change the implementation
  - replace the GC by a *predictable* allocator
    - use region-based memory management
  - automatically compute object lifetimes at compilation
    - undecidable problem
    - find a reasonable over-approximation

Outline

- Introduction
- Region-Based Memory management
- Pointer Interference Analysis
- Experimental results
- Conclusion

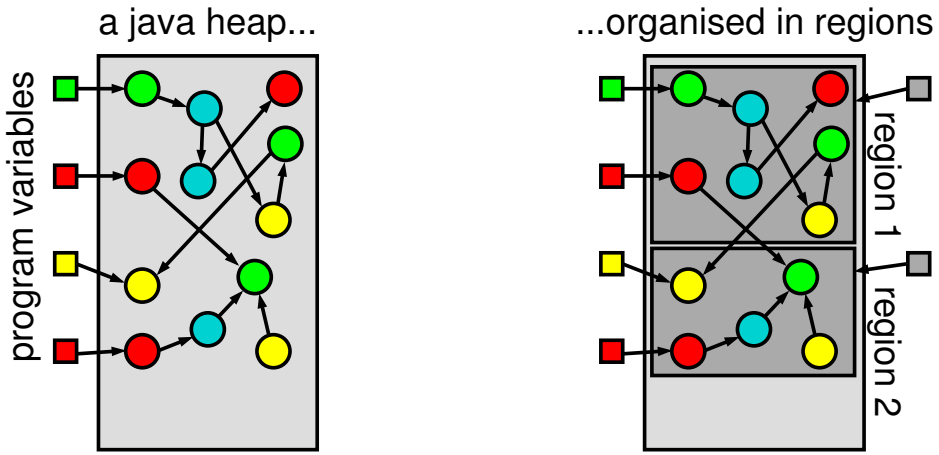
Memory management using regions



Objects in a region will share the same (physical) lifetime

- Benefits: a more *real-time*-compatible behaviour
  - objects allocated side by side
    - no fragmentation, constant time
  - region destroyed as a whole: predictable times
- Drawbacks: more difficult bookkeeping
  - object placement issue: who decides?
  - region destroyed as a whole: space overhead, risk of faults

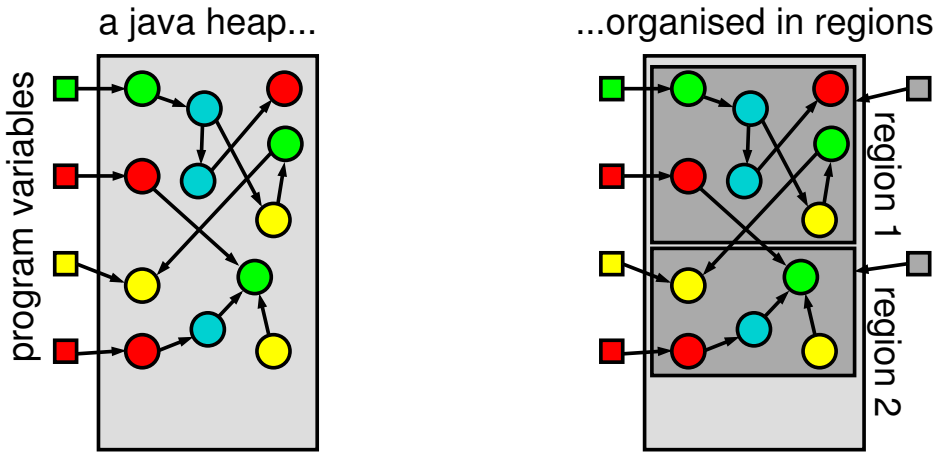
Memory management using regions



Objects in a region will share the same (physical) lifetime

- Benefits: a more *real-time*-compatible behaviour
  - objects allocated side by side
    - no fragmentation, constant time
  - region destroyed as a whole: predictable times
- Drawbacks: more difficult bookkeeping
  - object placement issue: who decides?
  - region destroyed as a whole: space overhead, risk of faults

Memory management using regions



Objects in a region will share the same (physical) lifetime

- Benefits: a more *real-time*-compatible behaviour
  - objects allocated side by side
    - no fragmentation, constant time
  - region destroyed as a whole: predictable times
- Drawbacks: more difficult bookkeeping
  - object placement issue: who decides?
  - region destroyed as a whole: space overhead, risk of faults

Outline

- Introduction
- Region-Based Memory management
- Pointer Interference Analysis
- Experimental results
- Conclusion

Region analysis and allocation policy

Hypothesis: Objects within the same data structure (i.e. connected together) will often have similar (logical) lifetimes

$\Longrightarrow$ one region for each data structure

- no inter-region pointer

Static analysis:

- identify variables that may point to connected objects

Allocation Policy:

- place objects so that each structure is grouped in a region

Region analysis and allocation policy

Hypothesis: Objects within the same data structure (i.e. connected together) will often have similar (logical) lifetimes

$\Longrightarrow$ one region for each data structure

- no inter-region pointer

Static analysis:

- identify variables that may point to connected objects

Allocation Policy:

- place objects so that each structure is grouped in a region

Example: pointer interference analysis

```
main()
{
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {
  this.data[this.index] = o;
  this.index ++;
 }
}
```

Example: pointer interference analysis

```
main()
{
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {
  this.data[this.index] = o;
  this.index ++;
 }
}
```

for all methods,

main()

<init>()

add()

Example: pointer interference analysis

```
main()
{
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {
  this.data[this.index] = o;
  this.index ++;
 }
}
```

identify local variables

main()

list *o1* *o2*

<init>()

add()

Example: pointer interference analysis

```
main()
{
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {
  this.data[this.index] = o;
  this.index ++;
 }
}
```

identify local variables

main()

list *o1* *o2*

<init>()

this *tmp*

add()

Example: pointer interference analysis

```
main()
{
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {
  this.data[this.index] = o;
  this.index ++;
 }
}
```

identify local variables

main()

list *o1* *o2*

<init>()

this *tmp*

add()

this *o*

Example: pointer interference analysis

```
main()
{
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {
  this.data[this.index] = o;
  this.index ++;
 }
}
```

local interference:

$v_1.\mathtt{f}=v_2 \Longrightarrow v_1 \sim v_2$

main()

list *o1* *o2*

<init>()

this $\sim$ *tmp*

add()

this *o*

Example: pointer interference analysis

```
main()
{
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {
  this.data[this.index] = o;
  this.index ++;
 }
}
```

local interference:

$v_1.\mathtt{f}=v_2 \Longrightarrow v_1 \sim v_2$

main()

list *o1* *o2*

<init>()

$this \sim tmp$

add()

$this \sim o$

Example: pointer interference analysis

```
main()
{
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {
  this.data[this.index] = o;
  this.index ++;
 }
}
```

interproc. interference:

$p_1 \sim p_2 \implies a_1 \sim a_2$

main()

$list \sim o1 \quad o2$

<init>()

$this \sim tmp$

add()

$this \sim o$

Example: pointer interference analysis

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```

results:

$main()$

$list \sim o1 \quad o2$

$<init>()$

$this \sim tmp$

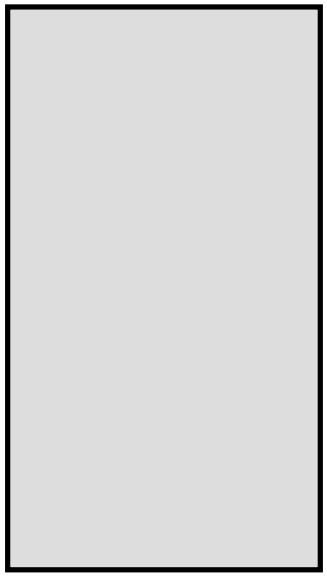
$add()$

$this \sim o$

Example: allocation policy

```
•
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

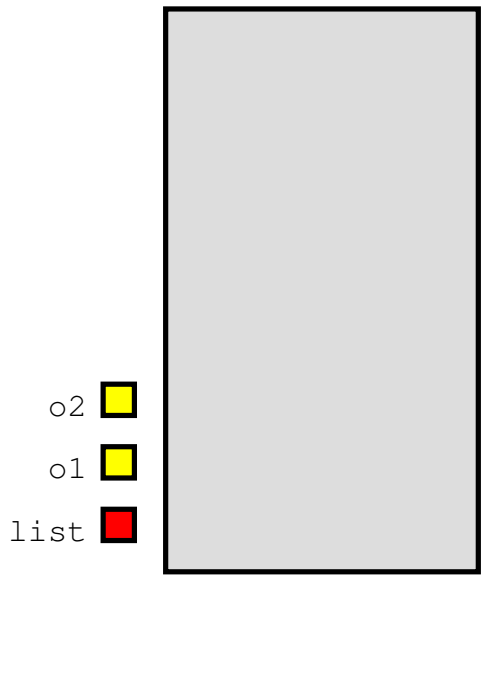
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list~o1   o2
●ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

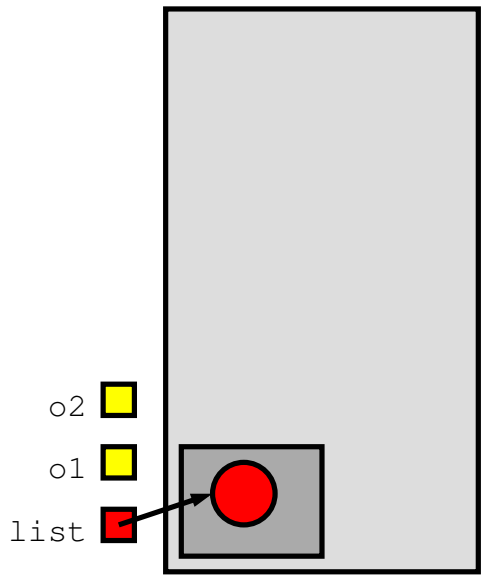
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this~tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this~o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list~o1   o2
 ArrayList list =
        new ArrayList;●
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

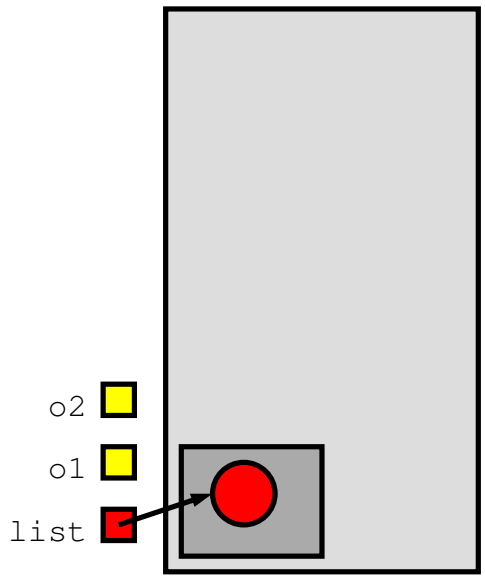
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this~tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this~o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
●list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

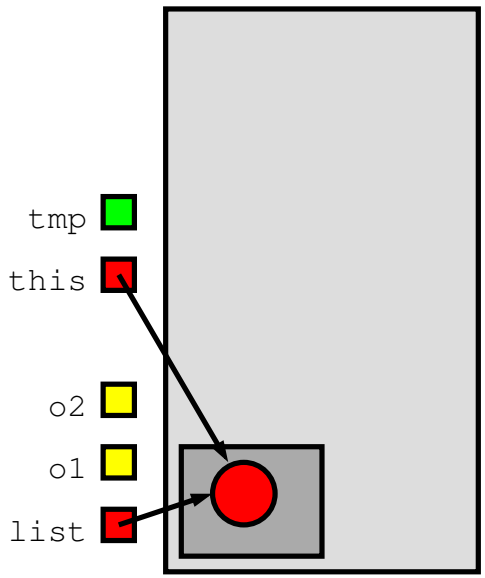
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
●list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

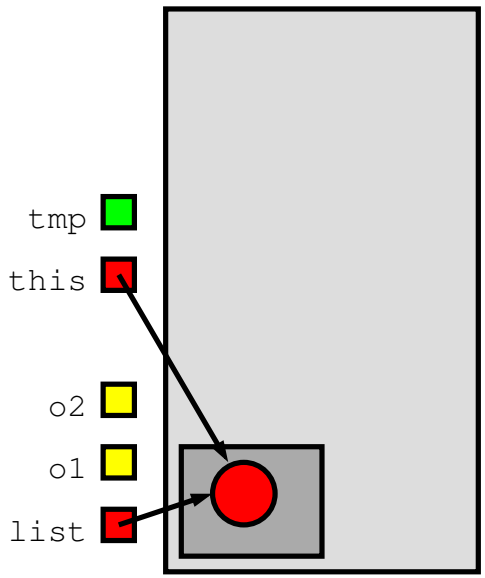
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
 ●this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
●list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

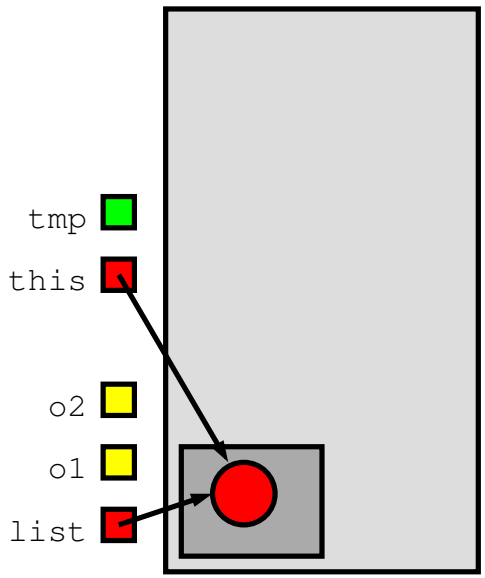
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;●
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
●list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

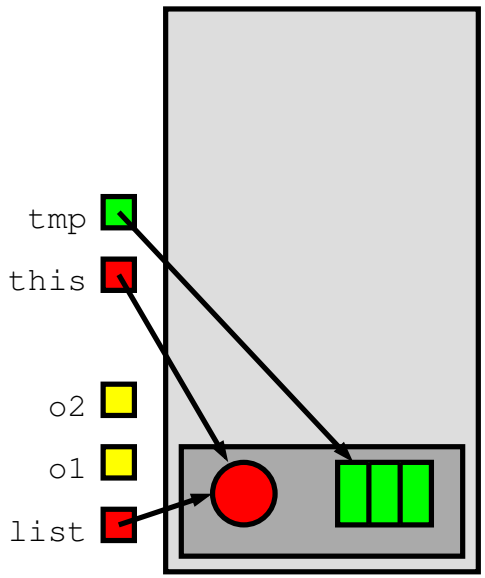
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;
 ●tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
●list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

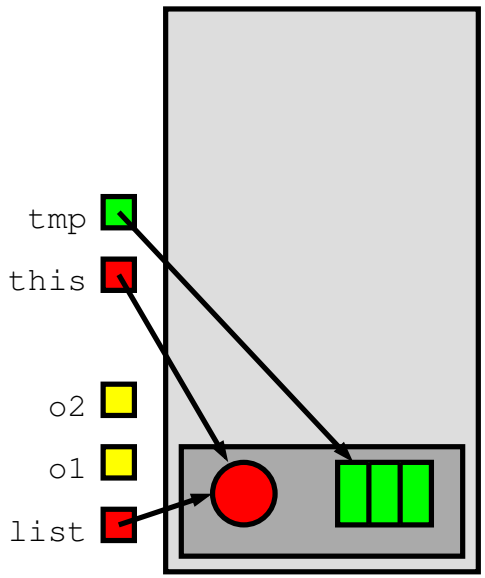
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;
  tmp = new Object[capacity];●
  this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
●list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

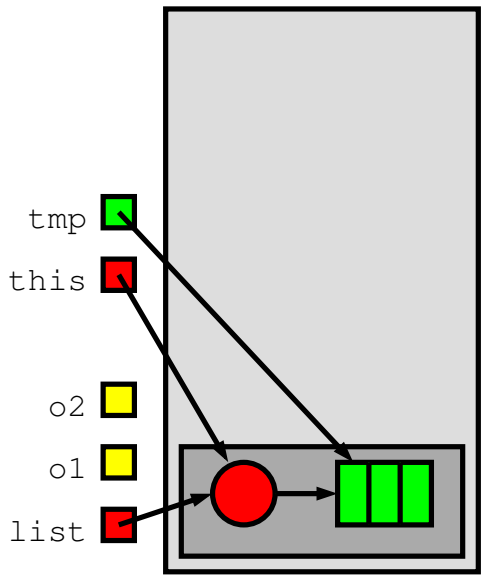
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;
  tmp = new Object[capacity];
 ●this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
●list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

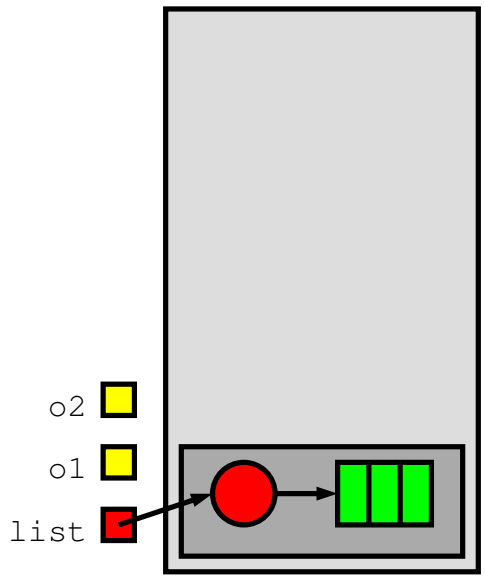
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;●
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);●

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

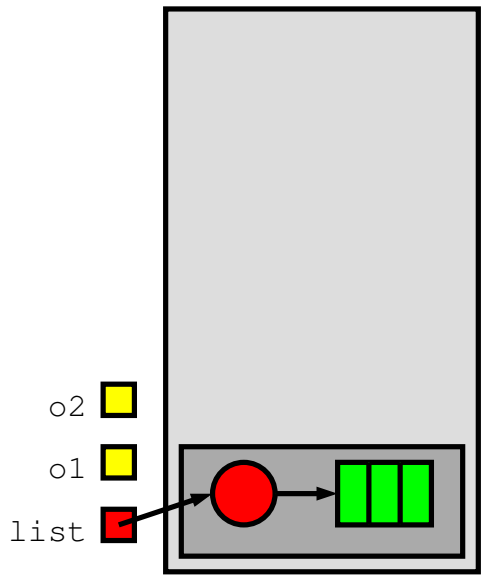
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);

●Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}
```

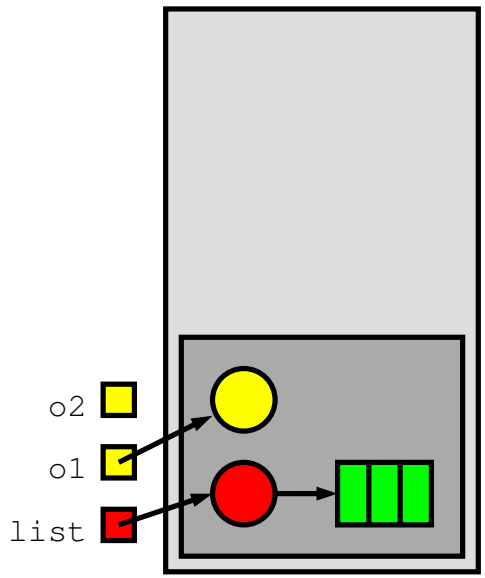
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;●
 Object o2=new Object;

 list.add(o1);
}
```

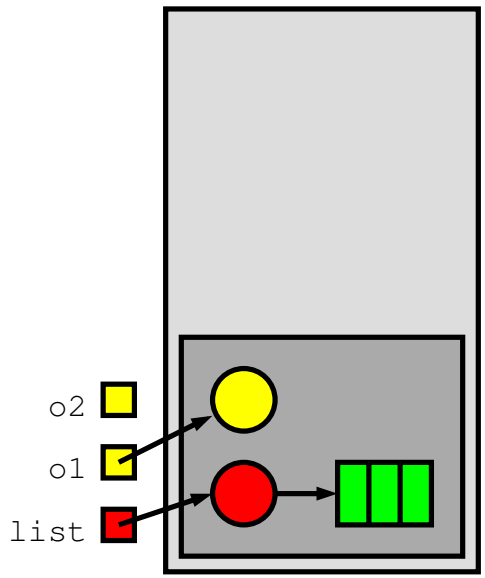
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list~o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
●Object o2=new Object;

 list.add(o1);
}
```

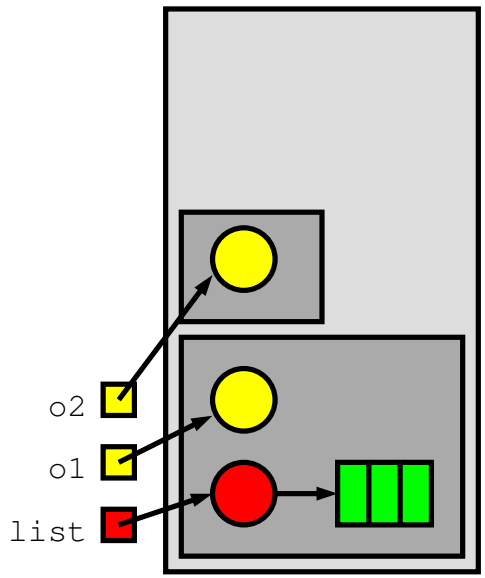
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this~tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this~o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list~o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;●

 list.add(o1);
}
```

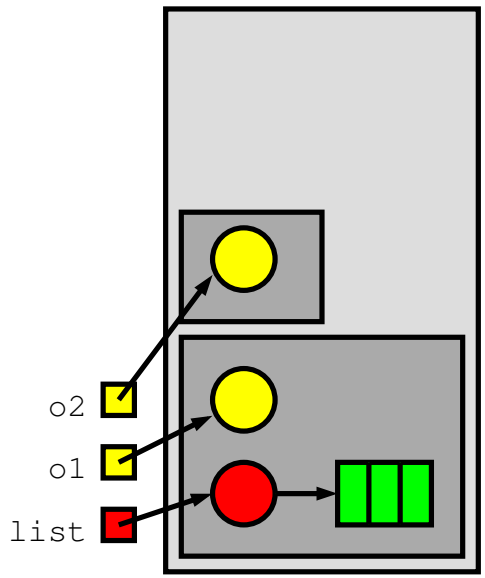
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this~tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this~o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

●list.add(o1);
}
```

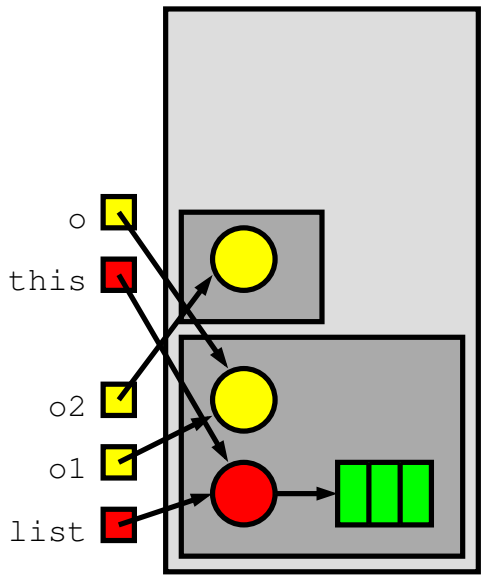
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list~o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

●list.add(o1);
}
```

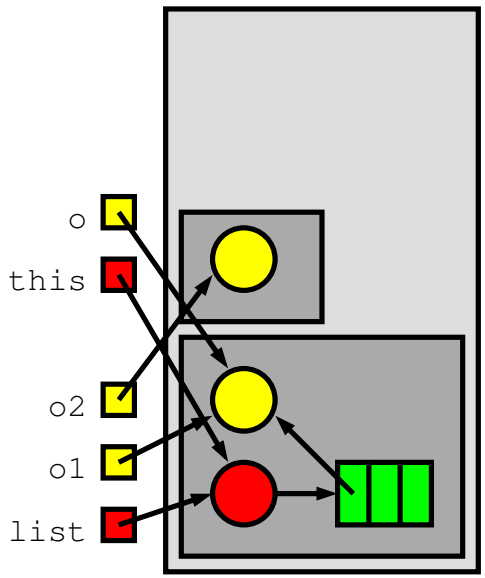
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this~tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this~o
 ●this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list~o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

●list.add(o1);
}
```

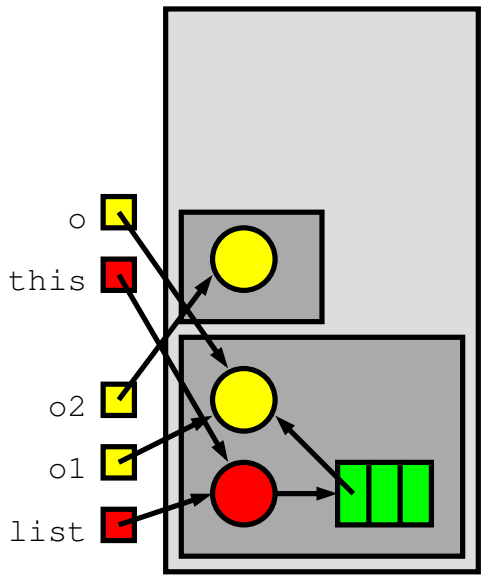
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this~tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this~o
  this.data[this.index] = o;●
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list~o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

●list.add(o1);
}
```

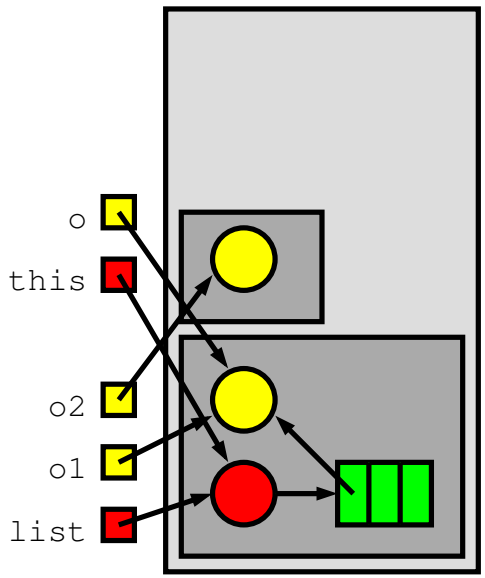
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this~tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this~o
  this.data[this.index] = o;
 ●this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

●list.add(o1);
}
```

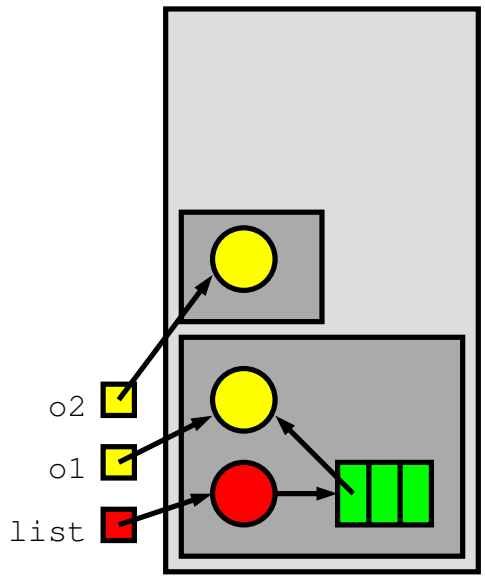
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;●
 }
}
```



Example: allocation policy

```
main()
{//list∼o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);●
}
```

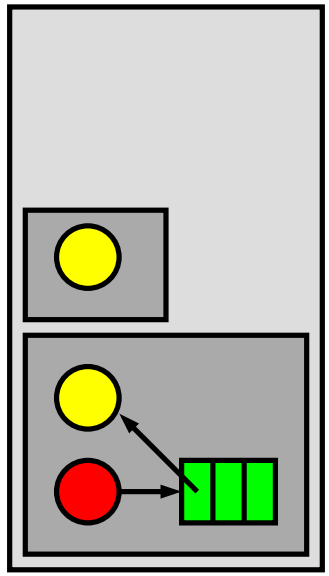
```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this∼tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this∼o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Example: allocation policy

```
main()
{//list~o1   o2
 ArrayList list =
        new ArrayList;
 list.<init>(3);

 Object o1=new Object;
 Object o2=new Object;

 list.add(o1);
}●
```

```
class ArrayList
{
 Object[] data;
 int index;

 <init>(int capacity)
 {//this~tmp
  this.index = 0;
  tmp = new Object[capacity];
  this.data = tmp;
 }

 void add(Object o)
 {//this~o
  this.data[this.index] = o;
  this.index ++;
 }
}
```



Outline

Experimental setup

Static analysis implemented with the Soot infrastructure

Memory manager implemented in the JITS virtual machine

JITS = *Java In The Small* (LIFL, France)

- a J2SE JavaOS for resource-constrained systems

Comparison of memory occupancy:

- with the default GC (mark & sweep)
- with static analysis + regions

JOlden benchmark suite

- lots of allocations
- different memory usage patterns

Experimental setup

Static analysis implemented with the Soot infrastructure

Memory manager implemented in the JITS virtual machine

JITS = *Java In The Small* (LIFL, France)

- a J2SE JavaOS for resource-constrained systems

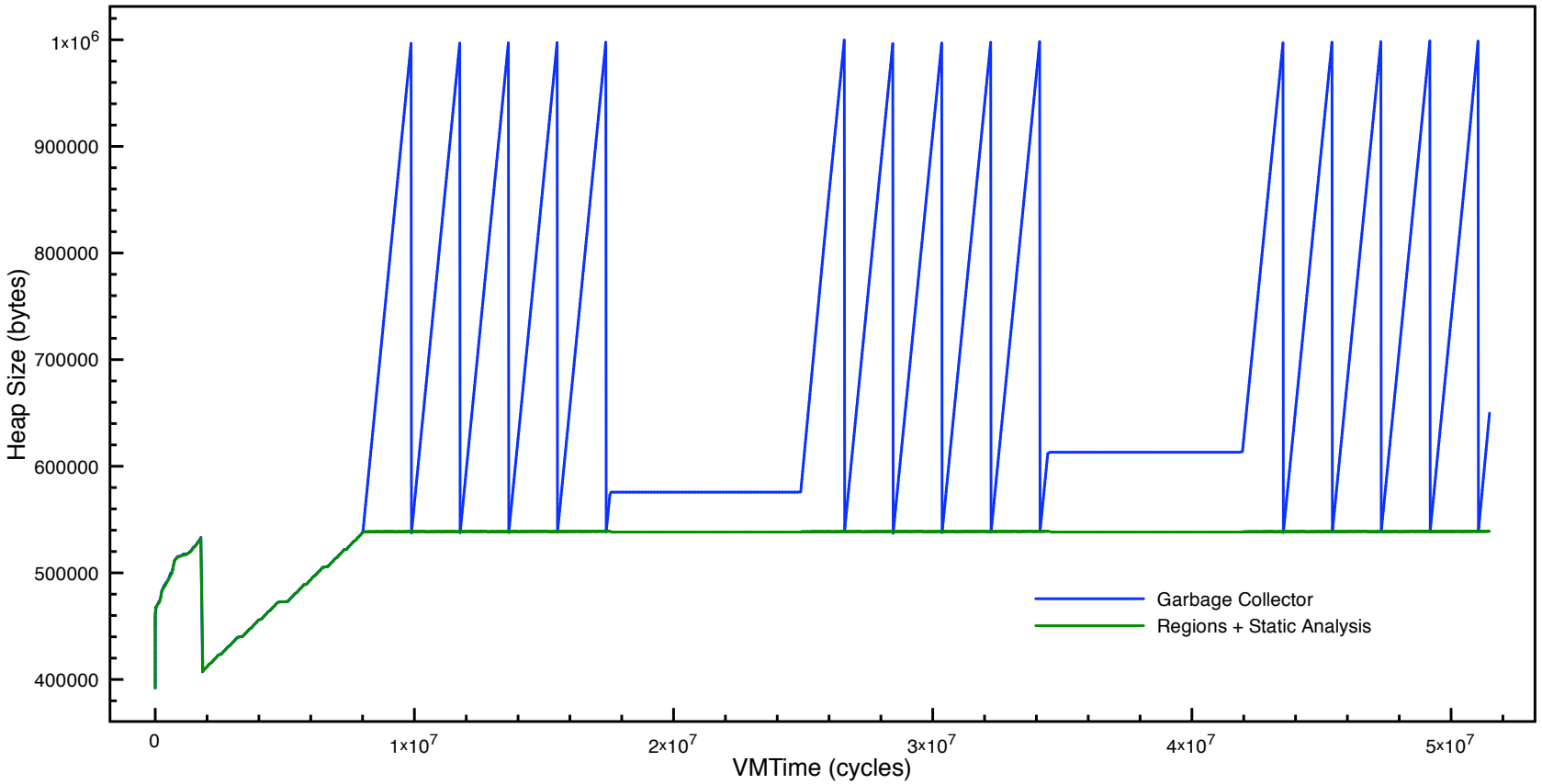
Comparison of memory occupancy:

- with the default GC (mark & sweep)
- with static analysis + regions

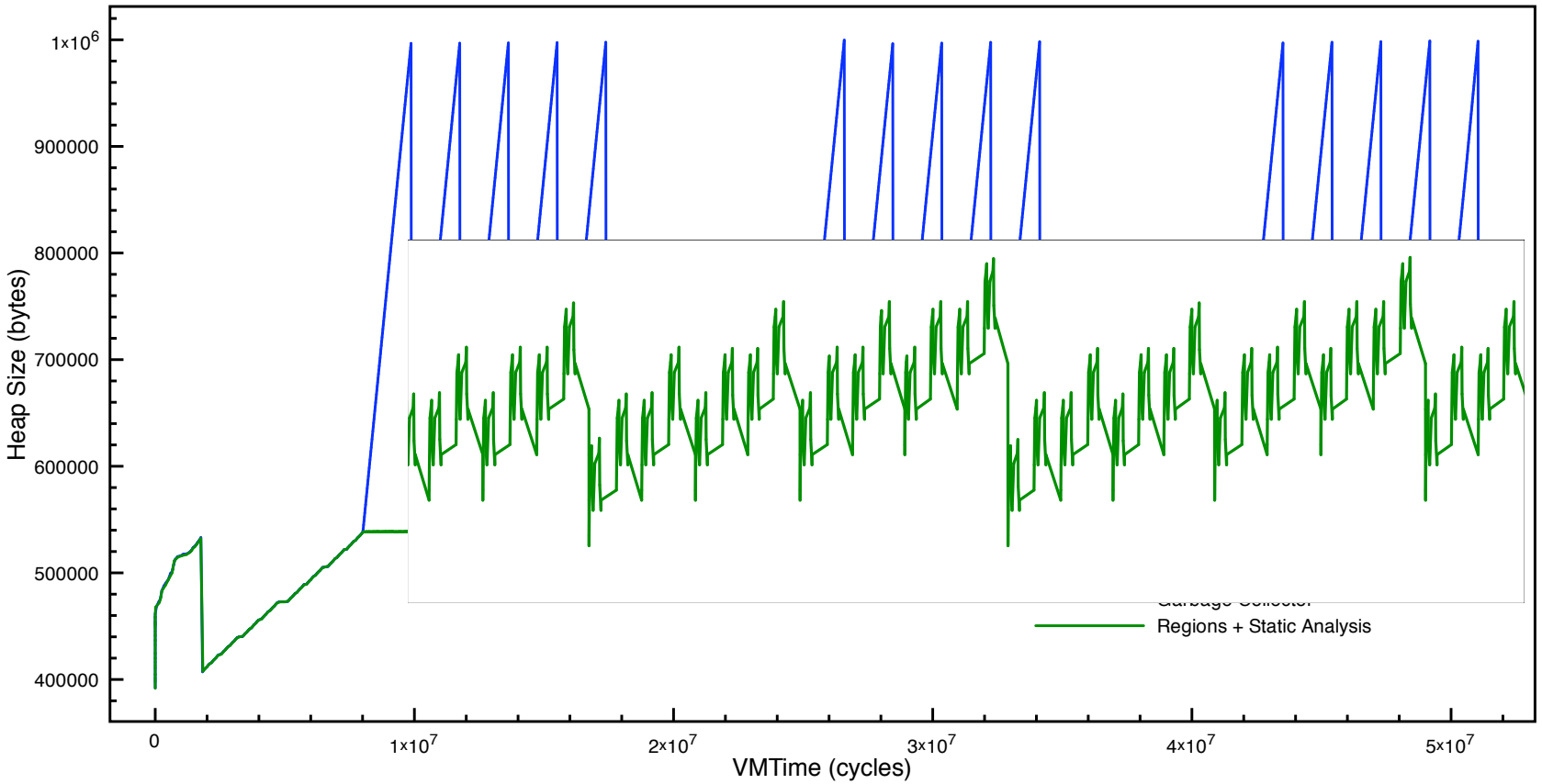
JOlden benchmark suite

- lots of allocations
- different memory usage patterns

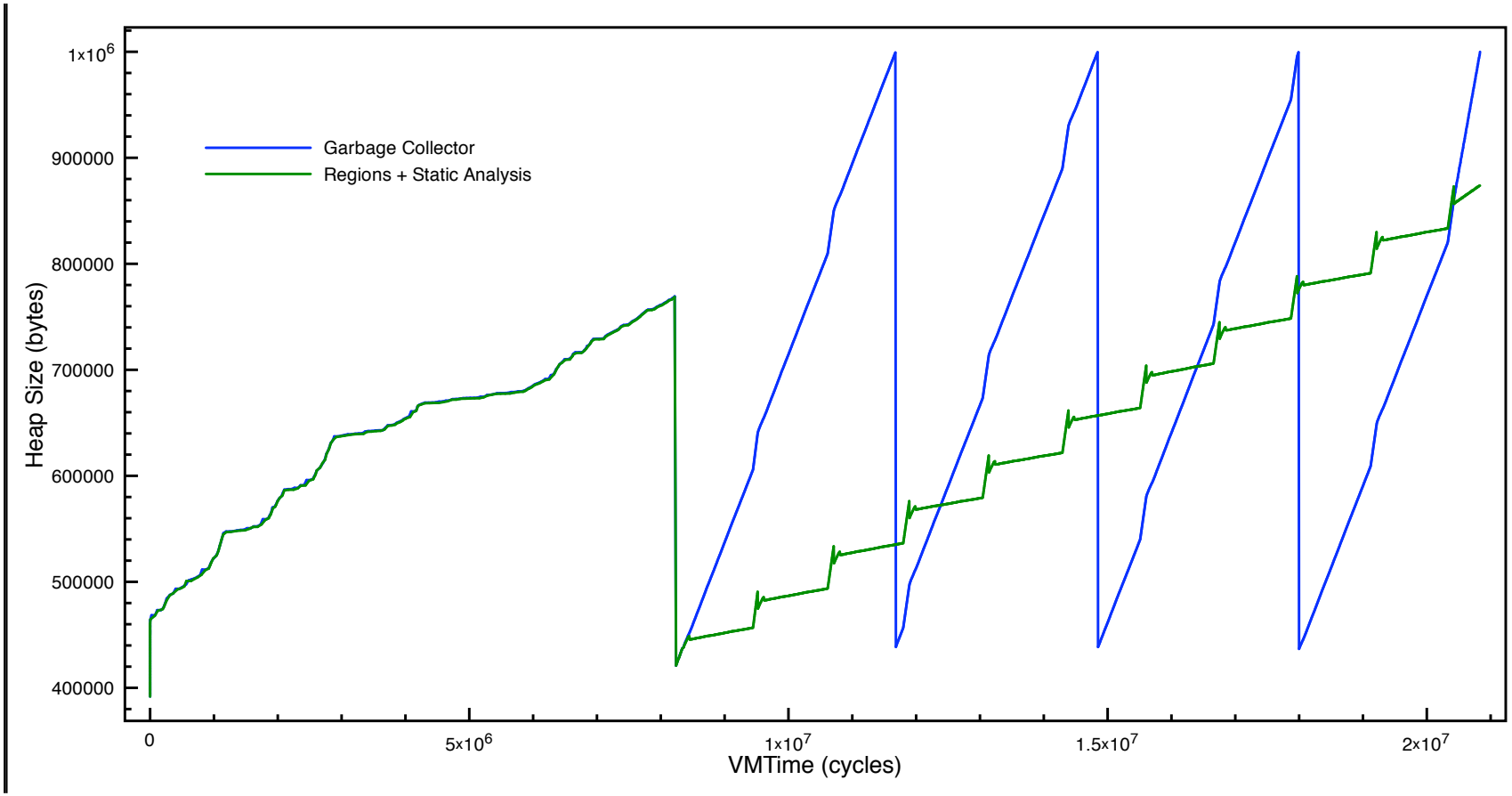
Experimental results (1)



Experimental results (1)



Experimental results (2)



Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```

Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```

Pointer interference analysis:

$$foo()$$

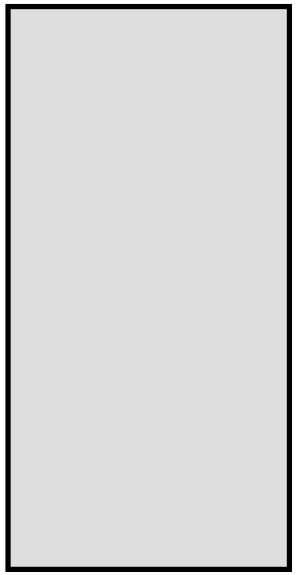
$$this \sim bar$$

$$main()$$

$$r$$

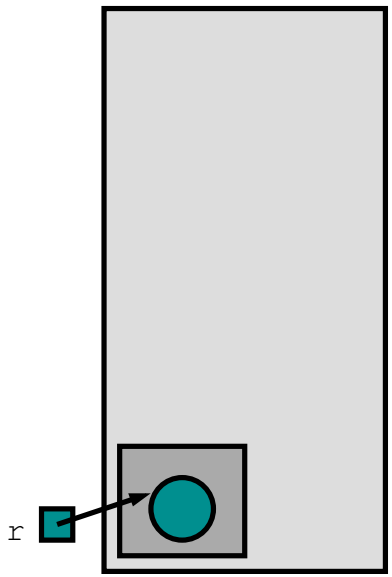
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



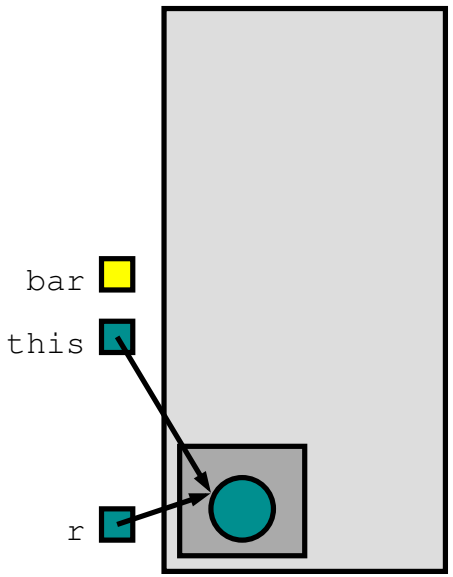
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



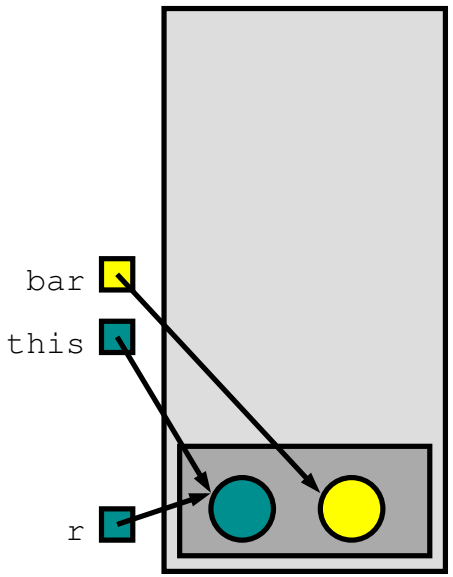
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



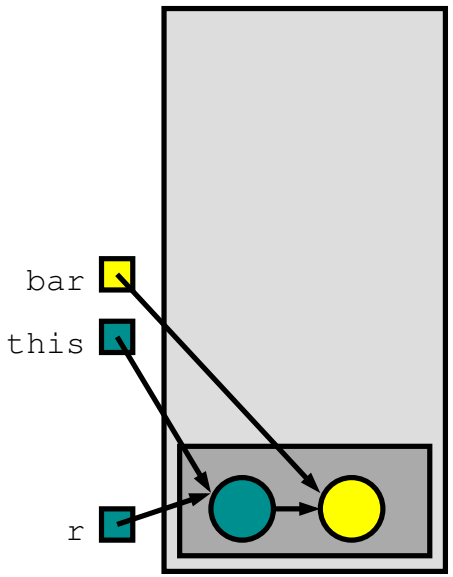
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



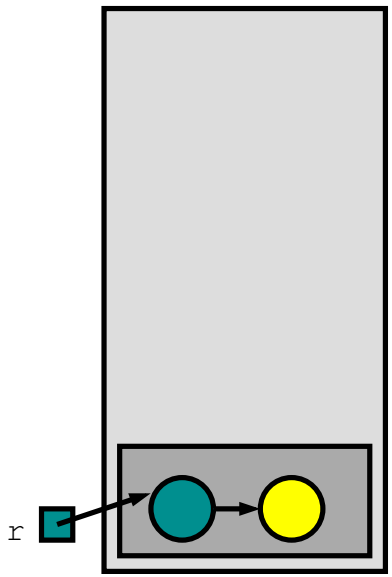
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



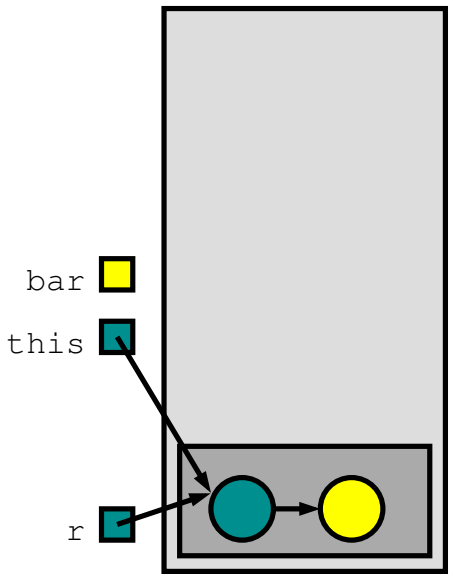
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



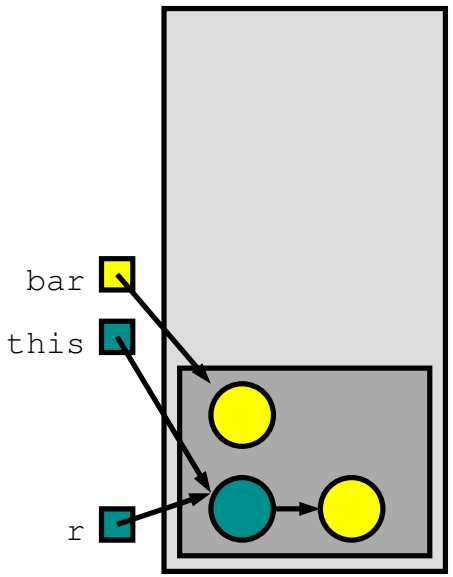
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



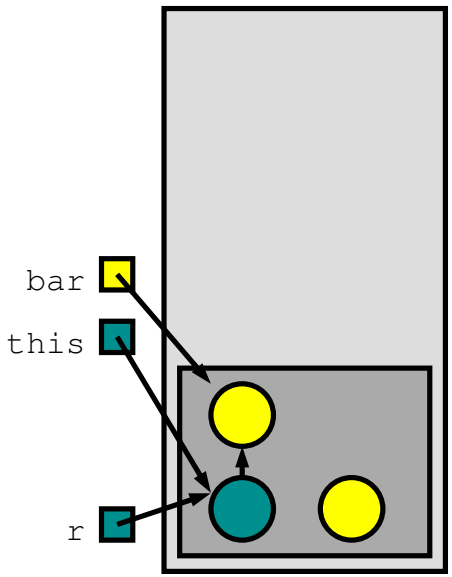
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



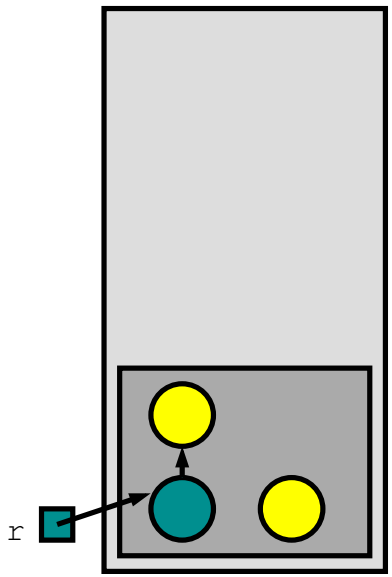
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



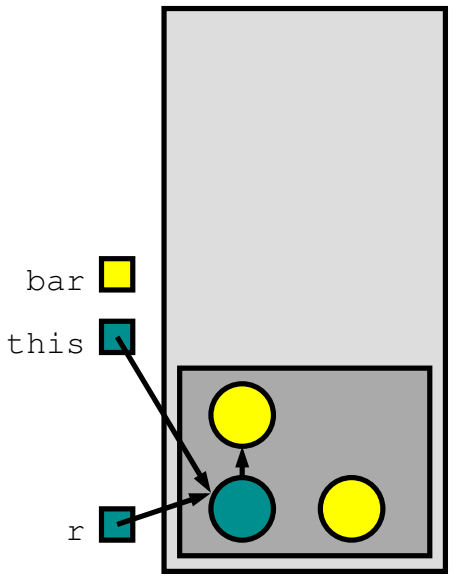
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



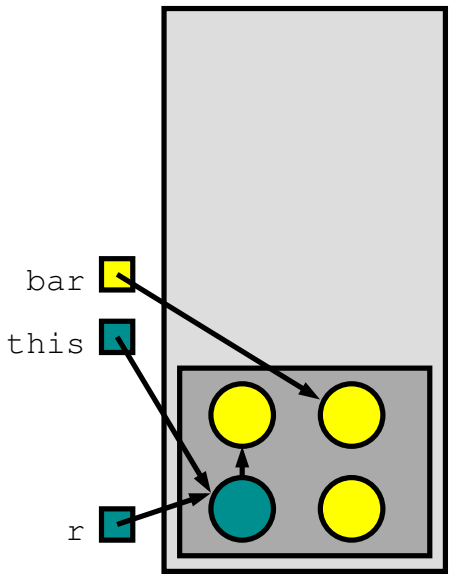
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



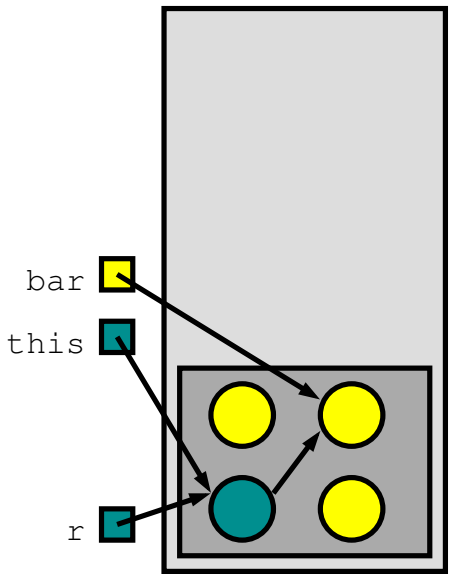
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



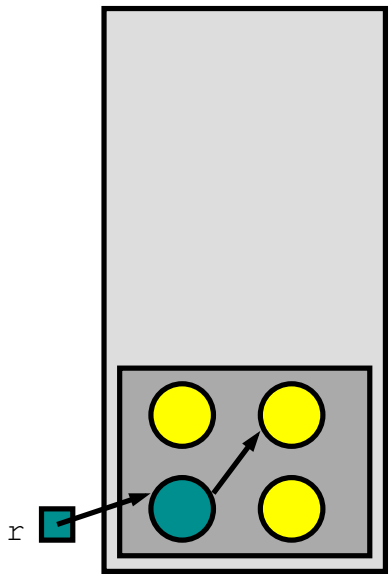
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



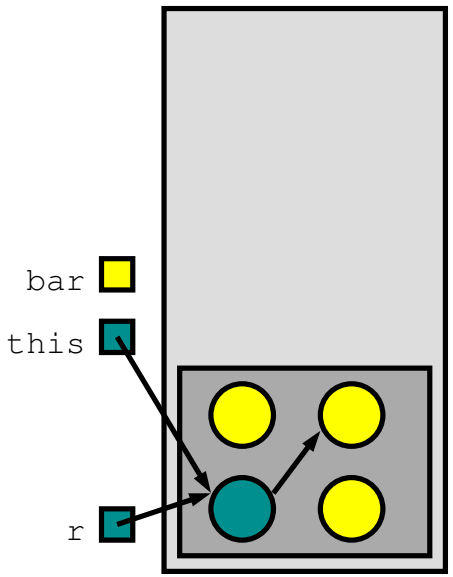
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



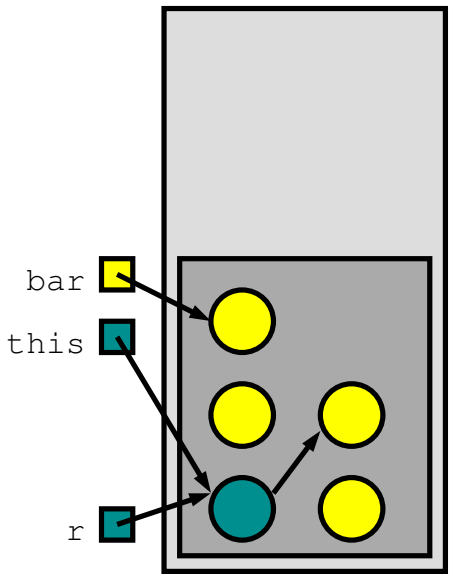
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



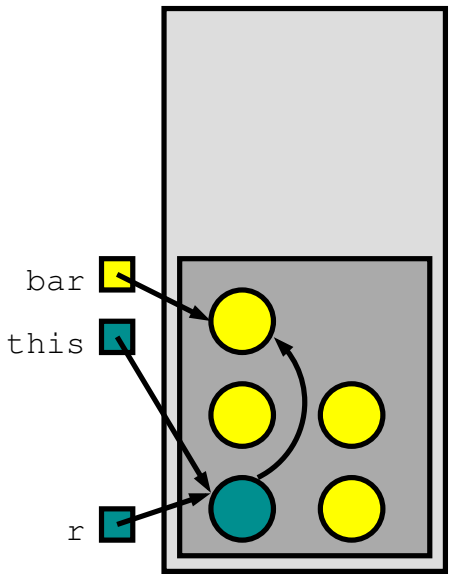
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



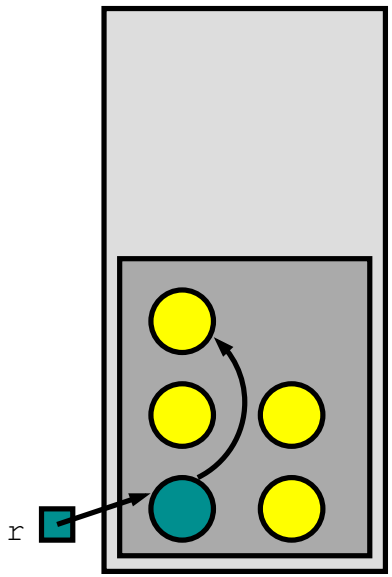
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



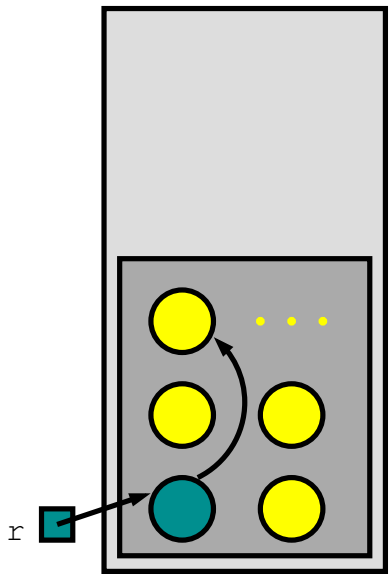
Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```



Region explosion syndrome

```
class RefObject
{
  Object f;

  foo()
  {//this~bar
    Object bar=new Object;
    this.f=bar;
  }
}

main()
{
  RefObject r=new RefObject();
  while(true)
  {
    r.foo();
  }
}
```

Region behaviour analysis:

search for bad paths in the call+interference graph...

...and report them to the programmer

Outline

Conclusion and perspectives

Results:

- a new region inference algorithm
- regions work fine for most programming patterns
- compile-time feedback to the programmer otherwise

Work in progress:

- improvement of the allocation policy
  - combination with a reference counting GC

Perspectives:

- extension to concurrency