Modular Reasoning About Global Variables and Their Initialization

João Pereira
Isaac van Bakel
Patricia Firlejczyk
Marco Eilers
Peter Müller

ETH zürich

Example: Lazy Initialization in Java

```
class Byte {
  public static Byte[] cache;









}
void client() {
  Byte zero = Byte.cache[0];
}
```

Example: Lazy Initialization in Java

```
class Byte {
  public static Byte[] cache;

  static {

    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }

  }
}
void client() {
  Byte zero = Byte.cache[0];
}
```

Example: Lazy Initialization in Java

```
class Byte {
  public static Byte[] cache;

  static {

    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }

  }
}

void client() {
  Byte zero = Byte.cache[0];
}
```

Example: Lazy Initialization in Java

```
class Byte {
  public static Byte[] cache;

  static {
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
  }
}

void client() {
  Byte zero = Byte.cache[0];
}
```

triggers

Example: Lazy Initialization in Java

```
class Byte {
  public static Byte[] cache;

  static {

    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }

  }
}
void client() {
  Byte zero = Byte.cache[0];
}
```

Example: Lazy Initialization in Java

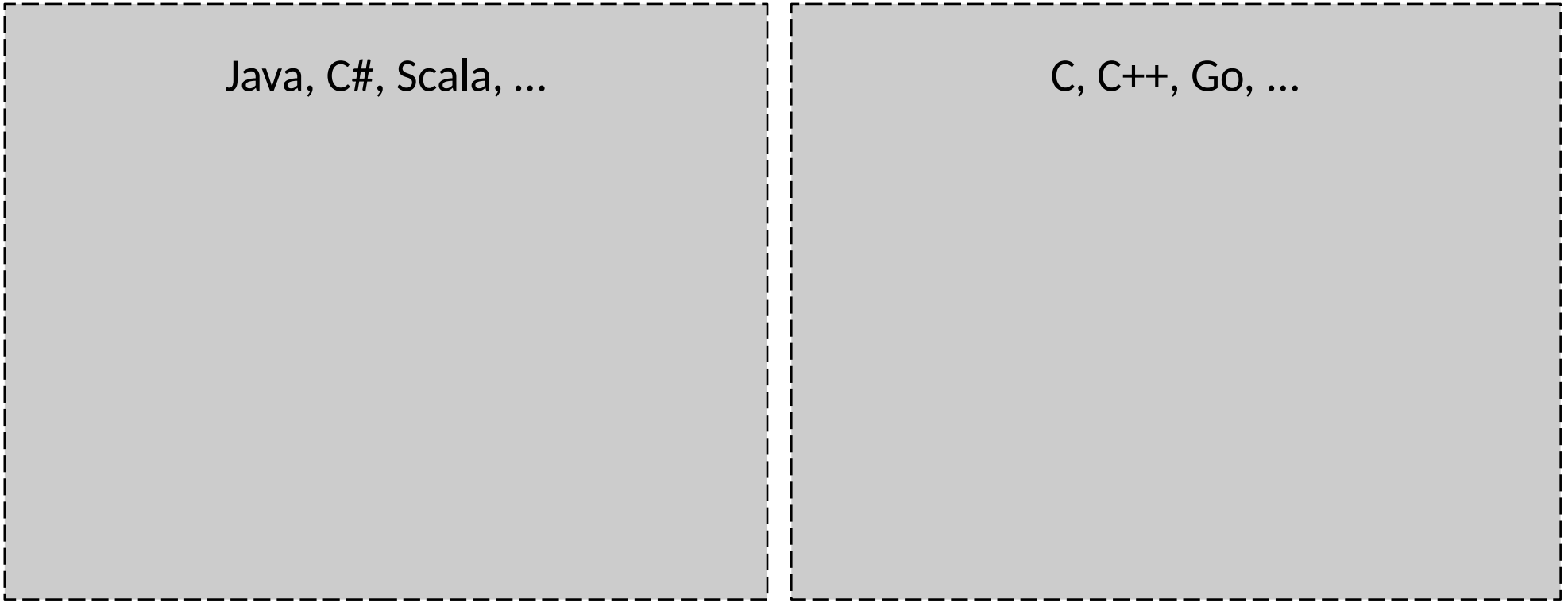
```
class Byte {
  public static Byte[] cache;

  static {
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
  }
}
void client() {
  Byte zero = Byte.cache[0];
}
```

returns

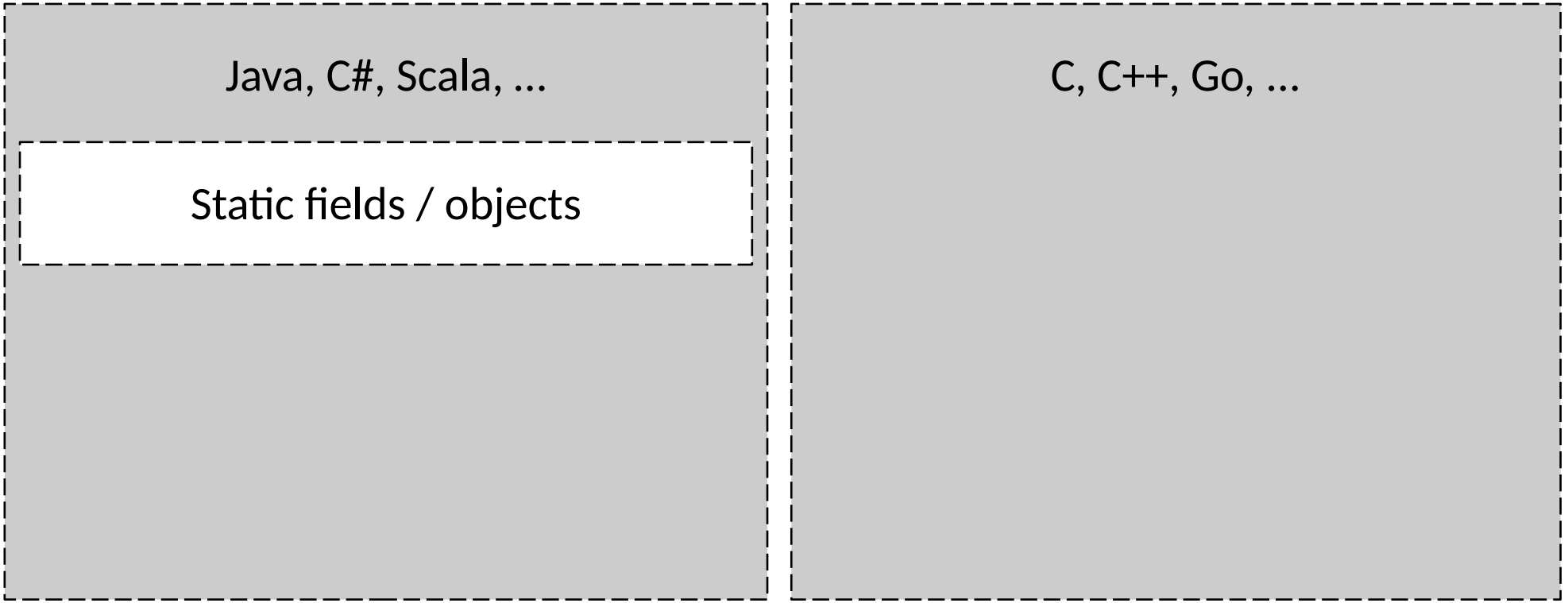
Example: Lazy Initialization in Java

```
class Byte {
  public static Byte[] cache;

  static {

    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }

  }
}

void client() {
  Byte zero = Byte.cache[0];
}
```

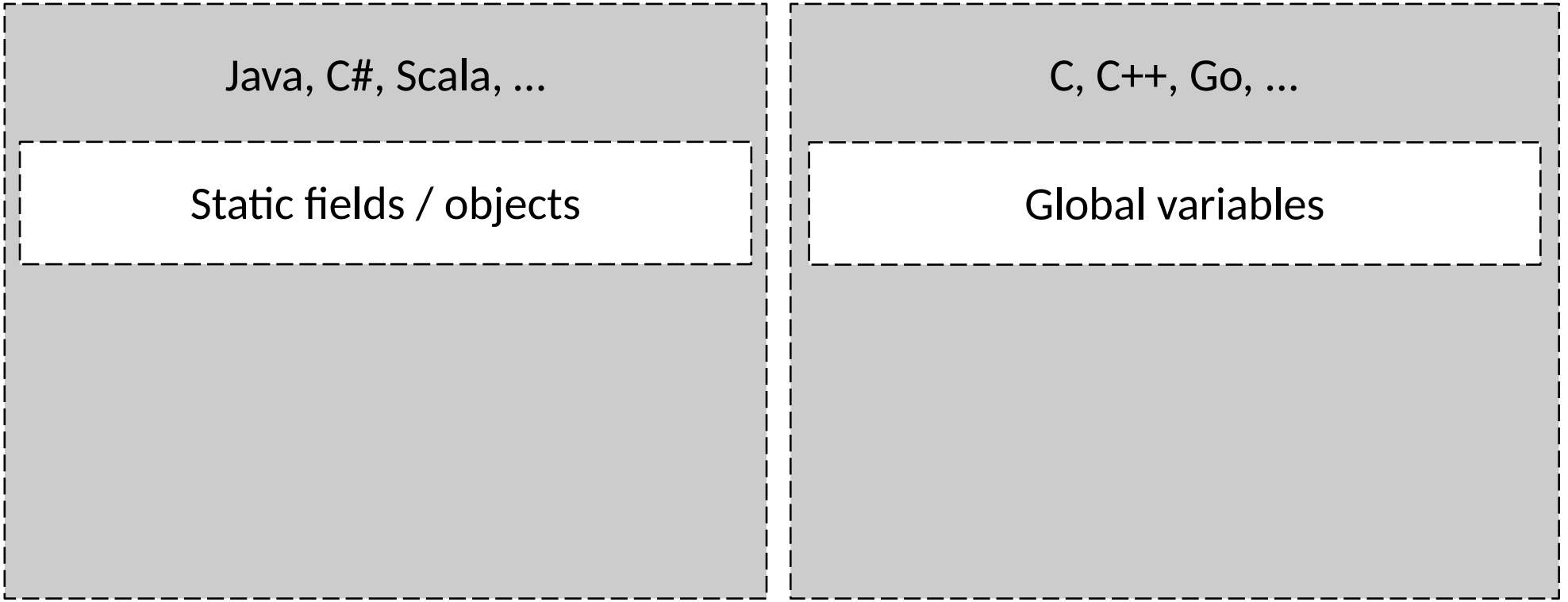
Initialization Semantics



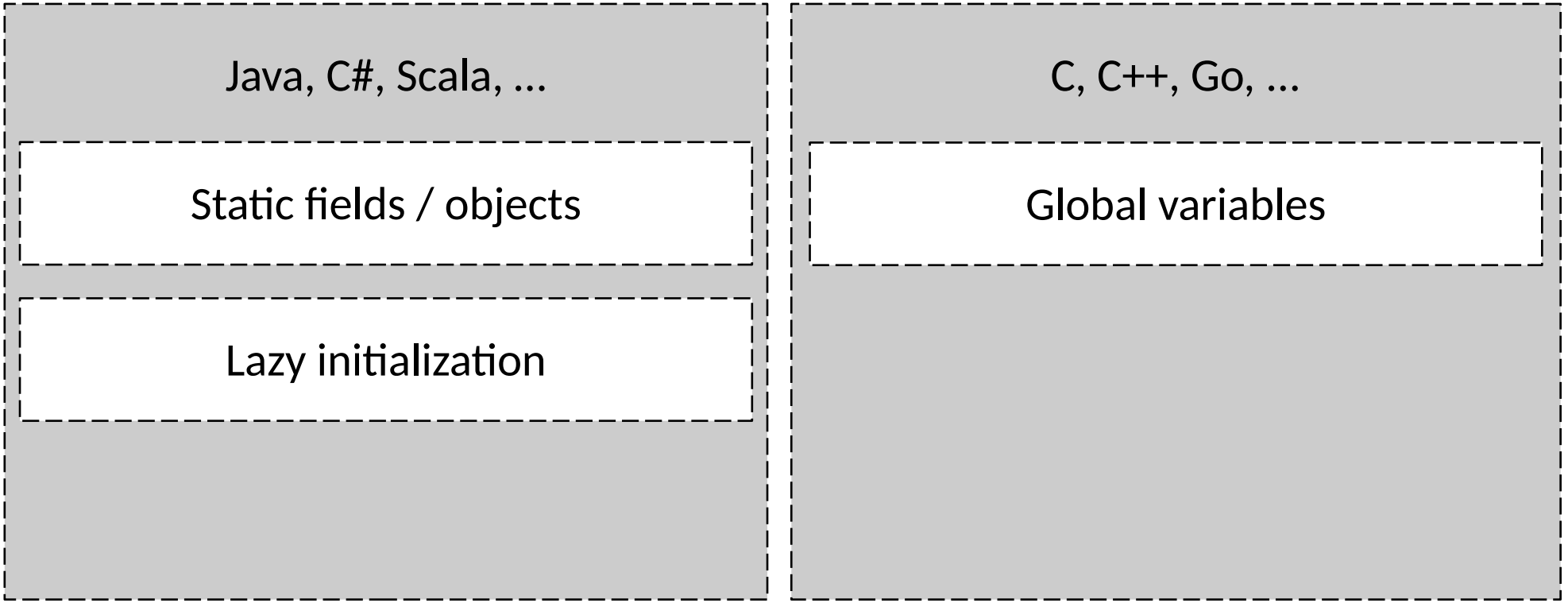
Initialization Semantics



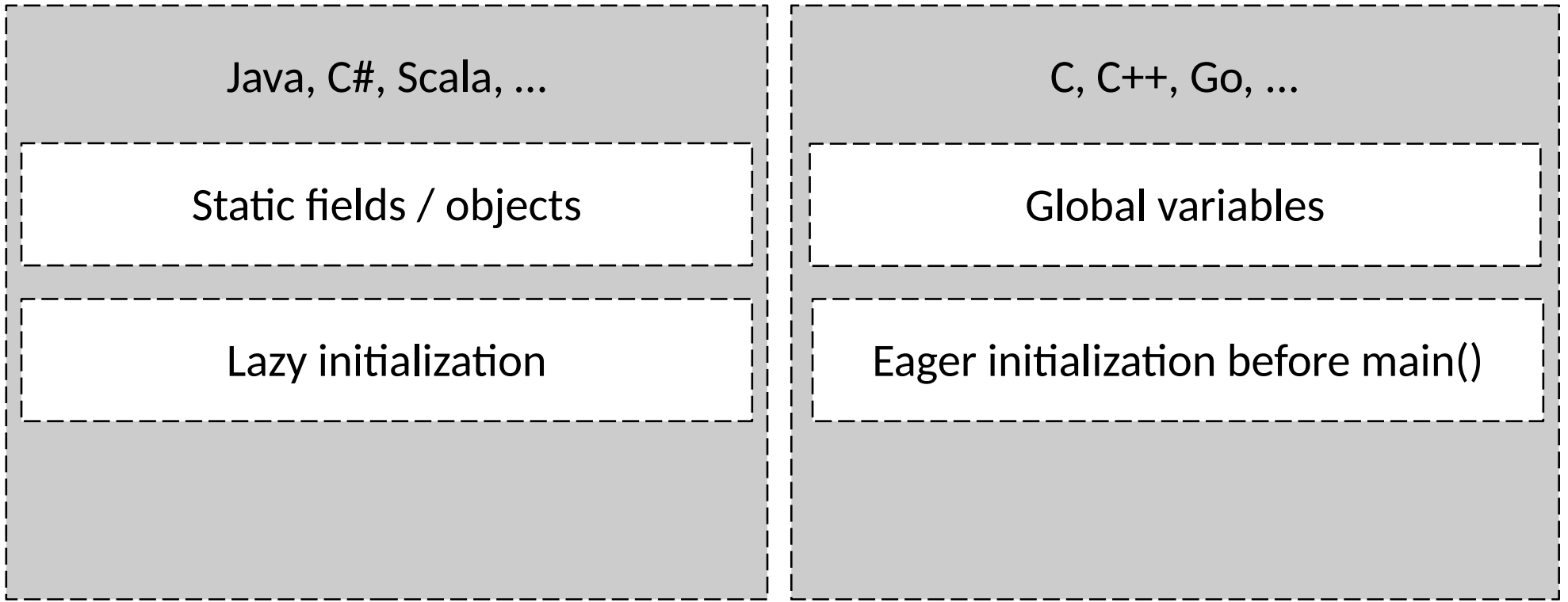
Initialization Semantics



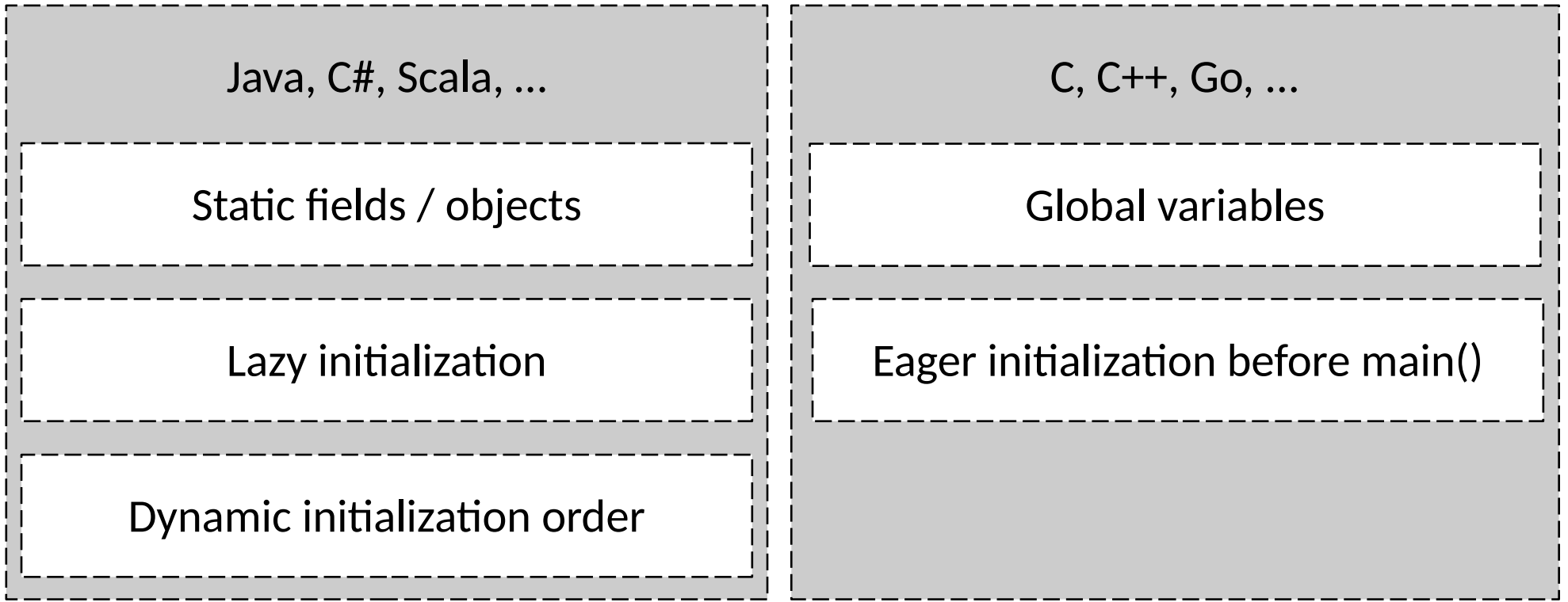
Initialization Semantics



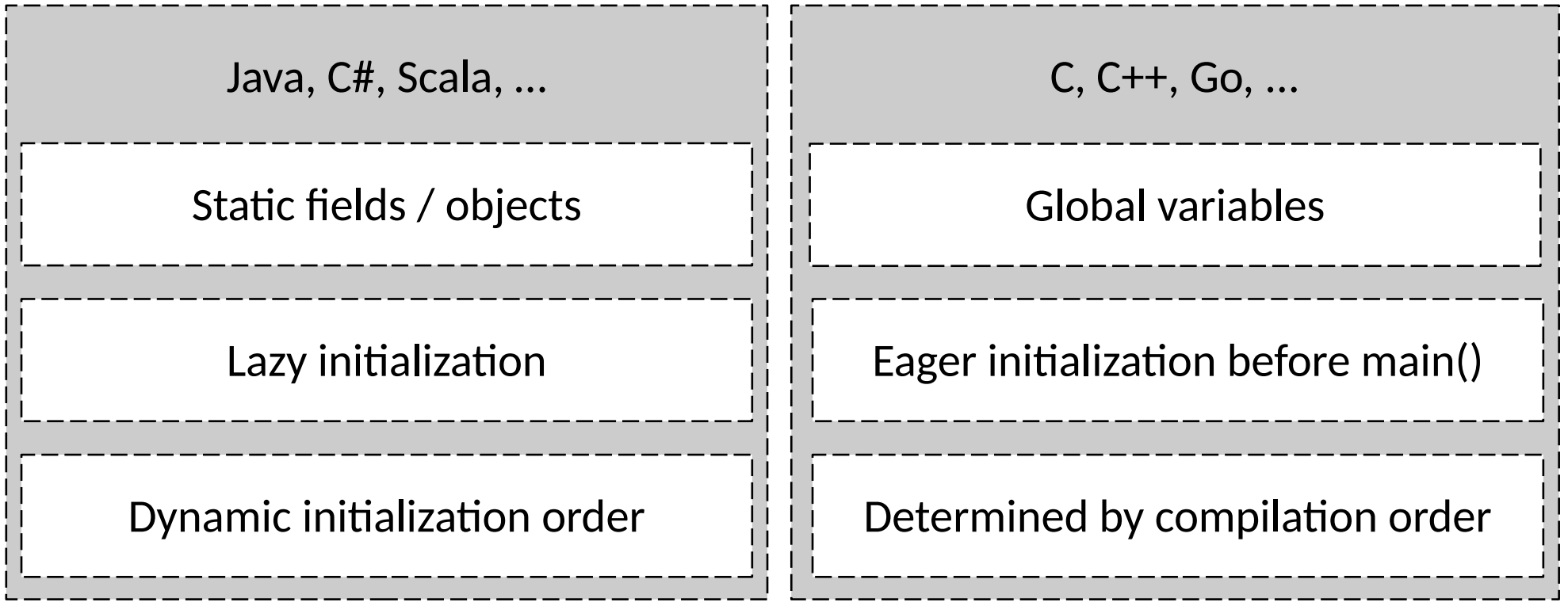
Initialization Semantics



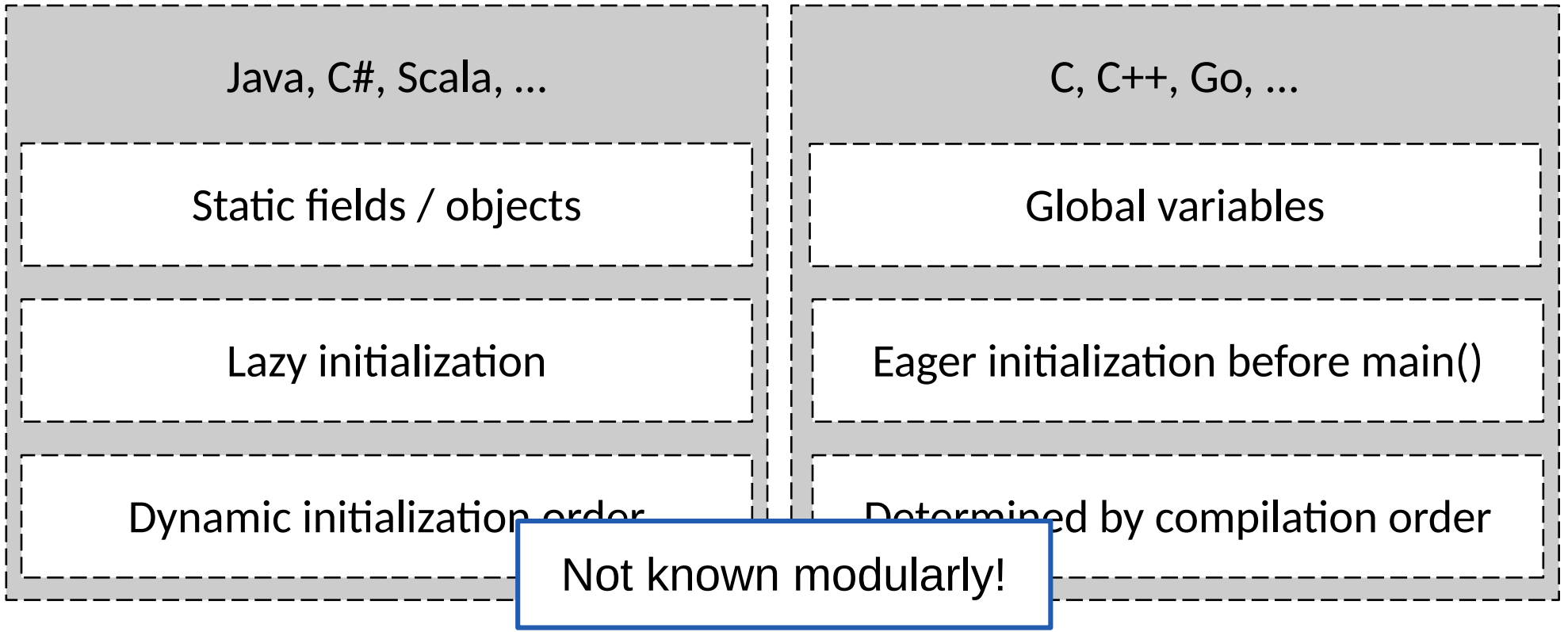
Initialization Semantics



Initialization Semantics



Initialization Semantics



Goal:

Modular,
language-independent reasoning
about global state
in concurrent separation logic

Approach: Module Invariants

Approach: Module Invariants

Initializer *establishes* invariant

Approach: Module Invariants

Initializer *establishes* invariant

```
class Byte {
  public static Byte[] cache;

  static {

    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
    { Byte.cache != null ∧ Byte.cache.length == 255 ∧ ... }
  }
}
```

Approach: Module Invariants

Initializer *establishes* invariant

Clients *use* invariant

```
class Byte {
  public static Byte[] cache;

  static {

    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
    { Byte.cache != null ∧ Byte.cache.length == 255 ∧ ... }
  }
}
```

Approach: Module Invariants

Initializer *establishes* invariant

Clients *use* invariant

```
class Byte {
  public static Byte[] cache;

  static {

    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
    { Byte.cache != null ∧ ... }
  }
}
```

```
class MyClass {
  void client() {
    { Byte.cache != null ∧ ... }
    Byte zero = Byte.cache[0];
  }
}
```

Key Challenge:

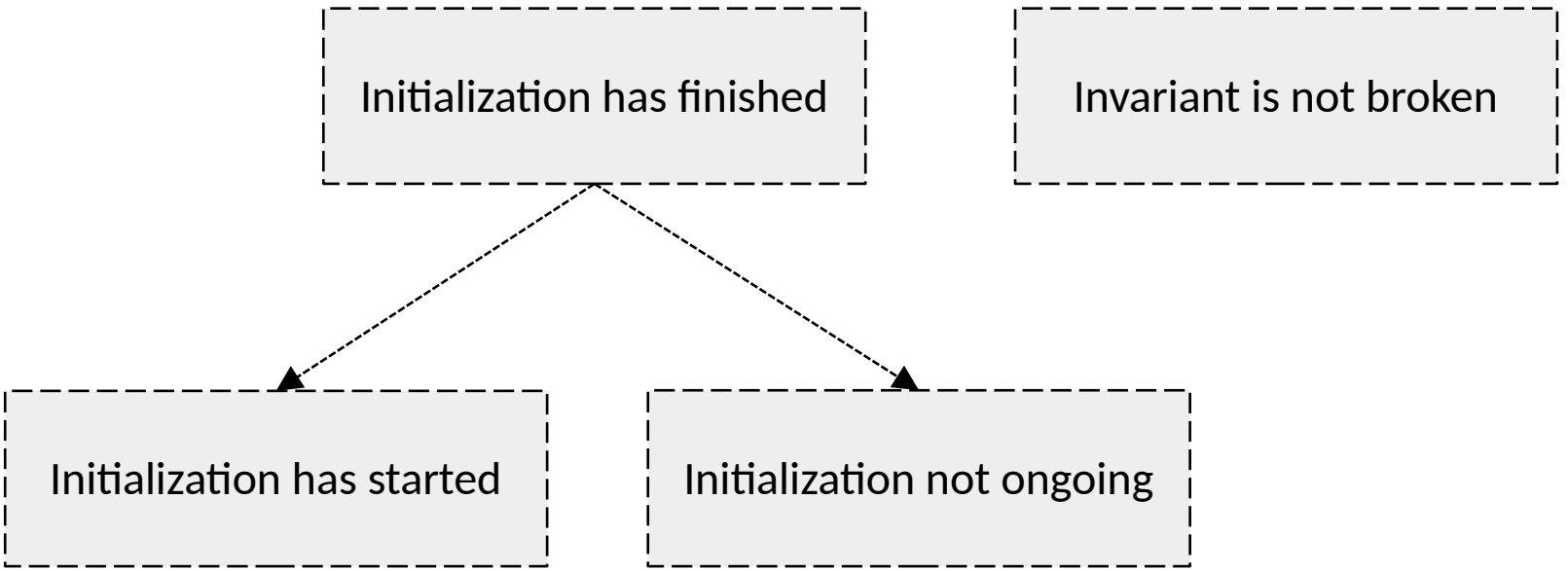
When is the invariant ***guaranteed*** to hold?

Proof Obligations

Initialization has finished

Invariant is not broken

Proof Obligations



Proof Obligations

Initialization has started

Initialization not ongoing

Invariant is not broken

Proof Obligations

Initialization has started

Proof Obligations

Initialization has started

```
class Byte {
  public static Byte[] cache;

  static {

    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
    { Byte.cache -> data * ... }
  }
}
```

Proof Obligations

Initialization has started

```
class Byte {
  public static Byte[] cache;

  static {

    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
    { Byte.cache -> data * ... }
  }
}
```

```
void client() {


  Byte zero = Byte.cache[0];
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

```
class Byte {
  public static Byte[] cache;

  static {

    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
    { Byte.cache -> data * ... }
  }
}
```

```
void client() {
  { Init_Byte }

  Byte zero = Byte.cache[0];
}
```

Proof Obligations

Initialization has started
$Init_M$-Fact

Initialization not ongoing

Proof Obligations

Initialization has started
$Init_M$-Fact

Initialization not ongoing

```
class Byte {
  public static Byte[] cache;

  static {
    new MyClass().client()
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }

  }
}
```

```
class MyClass {
  void client() {

    Byte zero = Byte.cache[0];
  }
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing

```
class Byte {
  public static Byte[] cache;

  static {
    new MyClass().client()
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }

  }
}
```

```
class MyClass {
  void client() {

    Byte zero = Byte.cache[0];
  }
}
```

Proof Obligations

Initialization has started
$Init_M$-Fact

Initialization not ongoing

```
class Byte {
  public static Byte[] cache;

  static {
    new MyClass().client()
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }

  }
}
```

```
class MyClass {
  void client() {

    Byte zero = Byte.cache[0];
  }
}
```

Proof Obligations

Initialization has started
$Init_M$-Fact

Initialization not ongoing

```
class Byte {
  public static Byte[] cache;

  static {
    new MyClass().client()
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }

  }
}
```

```
class MyClass {
  void client() {

    Byte zero = Byte.cache[0];
  }
}
```

Proof Obligations

Initialization has started
$Init_M$-Fact

Initialization not ongoing

```
class Byte {
  public static Byte[] cache;

  static {
    new MyClass().client()
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }

  }
}
```

```
class MyClass {
  void client() {
    { Byte.cache != null ∧ ... }
    Byte zero = Byte.cache[0];
  }
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing

```
class Byte {
  public static Byte[] cache;

  static {
    new MyClass().client()
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }

  }
}
```

```
class MyClass {
  void client() {
    { Byte.cache != null ∧ ... }
    Byte zero = Byte.cache[0];
  }
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

```
class Byte {
  public static Byte[] cache;

  static {
    new MyClass().client()
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }

  }
}
```

```
class MyClass {
  void client() {

    Byte zero = Byte.cache[0];
  }
}
```

Initialization Levels

```
// level 1
class Byte {
  public static Byte[] cache;

  // level 0
  static {


    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
  }
}
```

```
class MyClass {
  // level 0
  void client() {
    Byte zero = Byte.cache[0];
  }
}
```

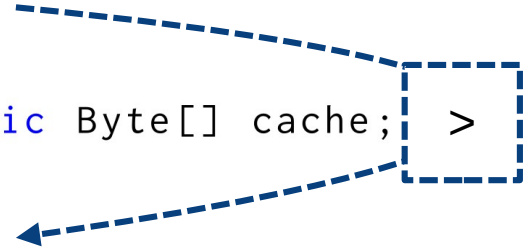
Initialization Levels

```
// level 1
class Byte {
  public static Byte[] cache;

  // level 0
  static {
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
  }
}
```

```
class MyClass {
  // level 0
  void client() {
    Byte zero = Byte.cache[0];
  }
}
```

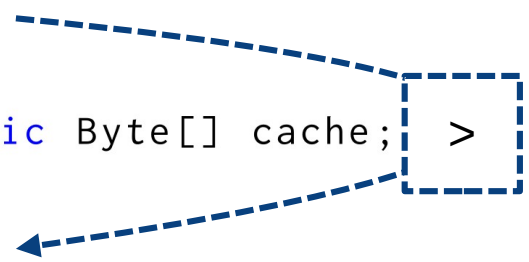
Initialization Levels

```
// level 1
class Byte {
  public static Byte[] cache;

  // level 0
  static {
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
  }
}
```




```
class MyClass {
  // level 0
  void client() {
    Byte zero = Byte.cache[0];
  }
}
```

Initialization Levels



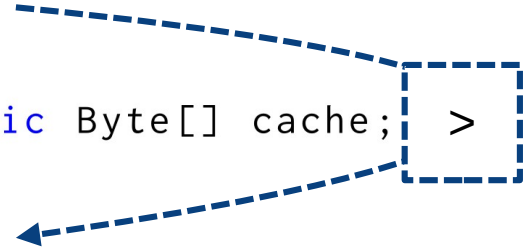
```
// level 1
class Byte {
  public static Byte[] cache;

  // level 0
  static {

    new MyClass().client()
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
  }
}
```

```
class MyClass {
  // level 0
  void client() {
    Byte zero = Byte.cache[0];
  }
}
```

Initialization Levels

Only decrease with calls

```
// level 1
class Byte {
  public static Byte[] cache;

  // level 0
  static {

    new MyClass().client()
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
  }
}
```




```
class MyClass {
  // level 0
  void client() {
    Byte zero = Byte.cache[0];
  }
}
```

Initialization Levels

Only decrease with calls

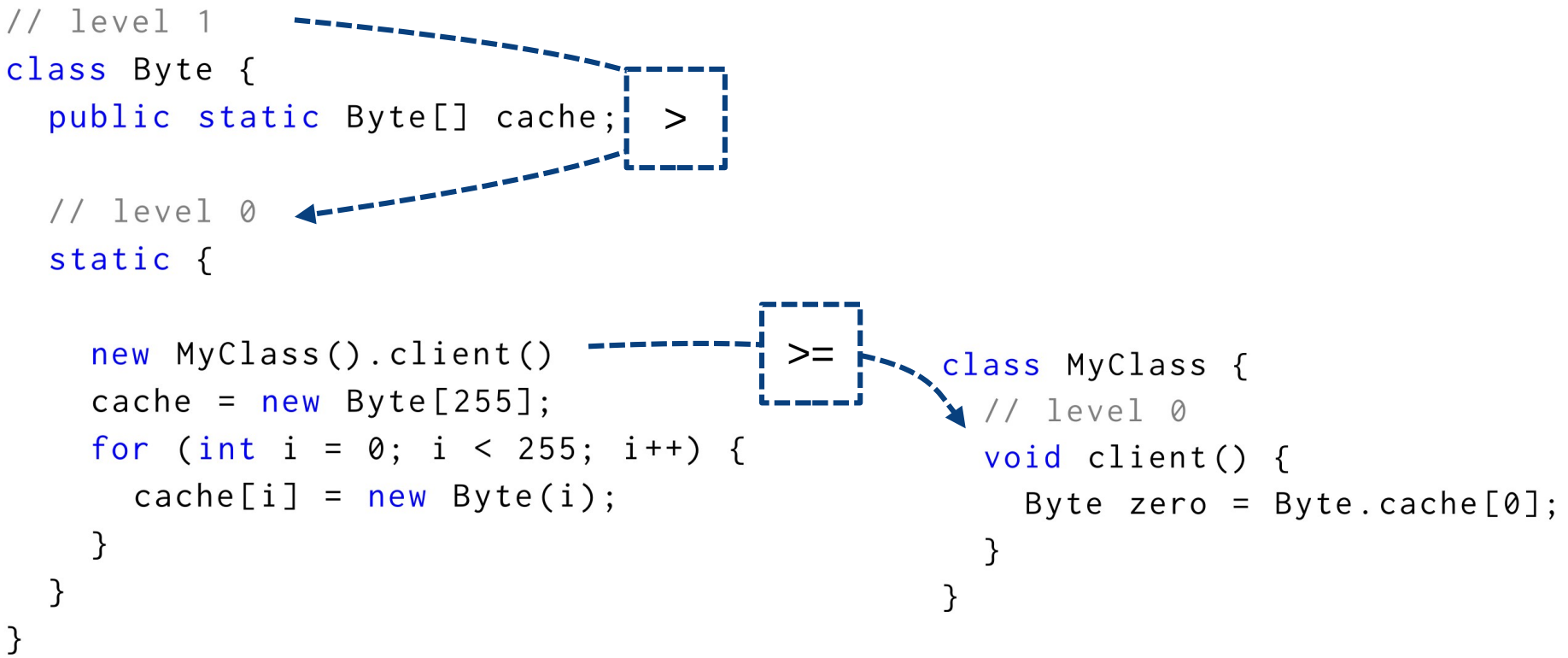
```
// level 1
class Byte {
  public static Byte[] cache;

  // level 0
  static {

    new MyClass().client()
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
  }
}
```

>

>=

```
class MyClass {
  // level 0
  void client() {
    Byte zero = Byte.cache[0];
  }
}
```

Initialization Levels



Initialization Levels

Use invariant with lower level only

```
// level 1
class Byte {
  public static Byte[] cache;

  // level 0
  static {

    new MyClass().client()
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
  }
}
```

>

>=

```
class MyClass {
  // level 0
  void client() {
    Byte zero = Byte.cache[0];
  }
}
```

Initialization Levels

Use invariant with lower level only

```
// level 1
class Byte {
  public static Byte[] cache;

  // level 0
  static {

    new MyClass().client()
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
  }
}
```

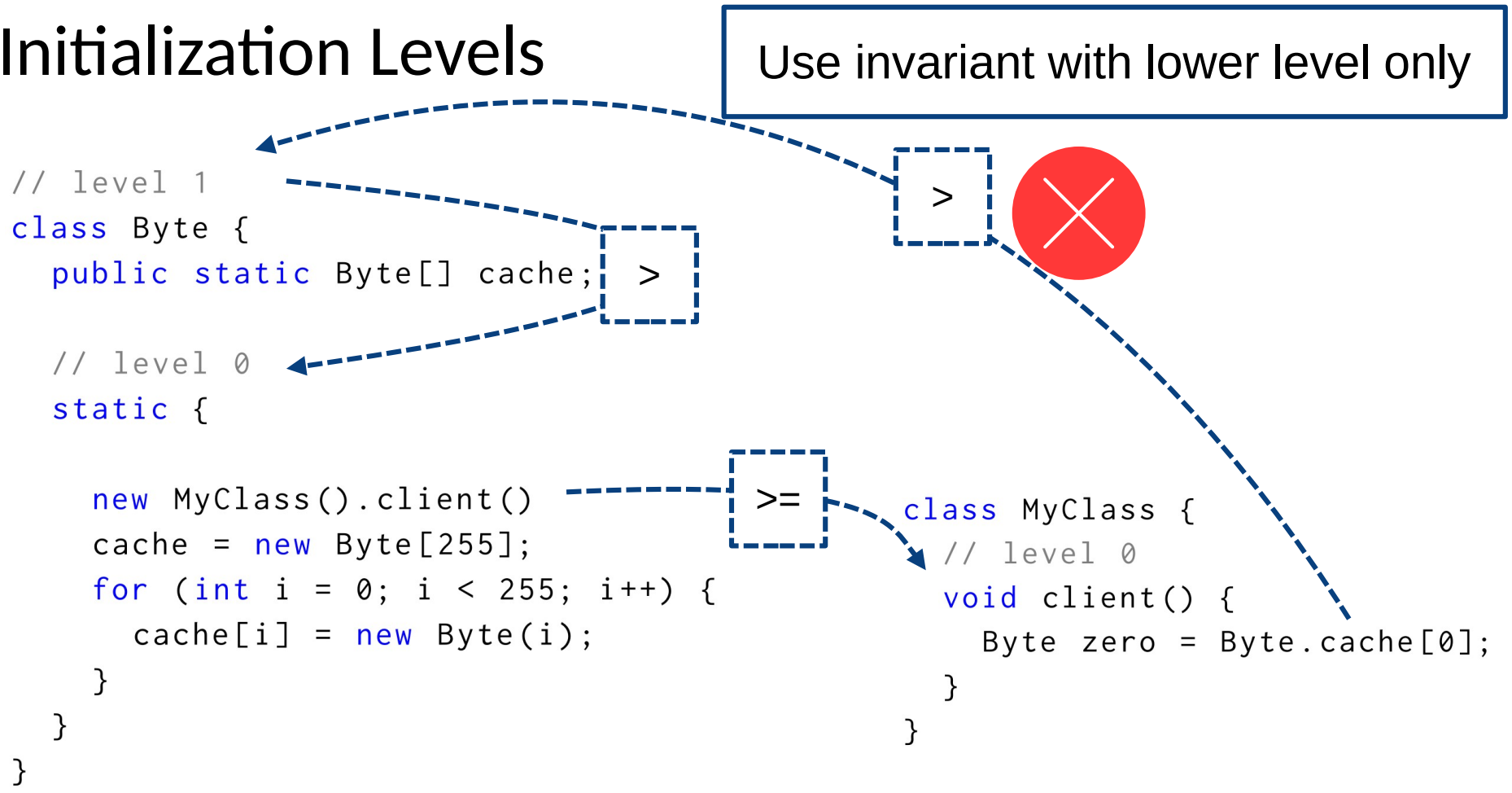
>

>

>=

```
class MyClass {
  // level 0
  void client() {
    Byte zero = Byte.cache[0];
  }
}
```

Initialization Levels



Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

```
void client() {
  { Init_Byte * L_cur > L_Byte }

  Byte zero = Byte.cache[0];
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken

```
class Disrupt {
  static {
    Byte.cache = null;
  }
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken
Control side effects

```
class Disrupt {
  static {
    Byte.cache = null;
  }
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken
Control side effects

```
class Disrupt {
  static {
    { emp }
    Byte.cache = null;
    { ... }
  }
}
```

```
class Byte {
  public static Byte[] cache;

  static {
    { Byte.cache -> _ }
    cache = new Byte[255];
    for (int i = 0; i < 255; i++) {
      cache[i] = new Byte(i);
    }
    { Byte.cache -> data * ... }
  }
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken
Control side effects

```
void clientCaller() {
  Byte.cache = null;
  client()
}
```

```
void client() {

  Byte zero = Byte.cache[0];
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken
Control side effects

```
void clientCaller() {
  Byte.cache = null;
  client()
}
```

```
void client() {
  { Byte.cache -> data * ... }
  Byte zero = Byte.cache[0];
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken
Control side effects
and *Module Tokens*

```
void clientCaller() {
  Byte.cache = null;
  client()
}
```

```
void client() {
  { Byte.cache -> data * ... }
  Byte zero = Byte.cache[0];
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken
Control side effects
and *Module Tokens*

```
class Main {

  { Token_MyClass * Token_Byte * ... }
  public static void main(String[] args) {
    new MyClass().client();
  }
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken
Control side effects
and *Module Tokens*

```
class Main {

  { Token_MyClass * Token_Byte * ... }
  public static void main(String[] args) {
    new MyClass().client();
  }
}
```

```
class MyClass {

  void client() {


    Byte zero = Byte.cache[0];


  }
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken
Control side effects
and *Module Tokens*

```
class Main {

  { Token_MyClass * Token_Byte * ... }
  public static void main(String[] args) {
    new MyClass().client();
  }
}
```

```
class MyClass {
  { Token_Byte }
  void client() {


    Byte zero = Byte.cache[0];


  }
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken
Control side effects
and *Module Tokens*

```
class Main {

  { Token_MyClass * Token_Byte * ... }
  public static void main(String[] args) {
    new MyClass().client();
  }
}
```

```
class MyClass {
  { Token_Byte }
  void client() {
    { Init_Byte * L_cur > L_Byte * Token_Byte }

    Byte zero = Byte.cache[0];


  }
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken
Control side effects
and *Module Tokens*

```
class Main {

  { Token_MyClass * Token_Byte * ... }
  public static void main(String[] args) {
    new MyClass().client();
  }
}
```

```
class MyClass {
  { Token_Byte }
  void client() {
    { Init_Byte * L_cur > L_Byte * Token_Byte }
      { cache -> data * ... }
    Byte zero = Byte.cache[0];


  }
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken
Control side effects
and *Module Tokens*

```
class Main {

  { Token_MyClass * Token_Byte * ... }
  public static void main(String[] args) {
    new MyClass().client();
  }
}
```

```
class MyClass {
  { Token_Byte }
  void client() {
    { Init_Byte * L_cur > L_Byte * Token_Byte }
      { cache -> data * ... }
    Byte zero = Byte.cache[0];
      { cache -> data * ... }

  }
}
```

Proof Obligations

Initialization has started
Init$_M$-Fact

Initialization not ongoing
Initialization Levels

Invariant is not broken
Control side effects
and *Module Tokens*

```
class Main {

  { Token_MyClass * Token_Byte * ... }
  public static void main(String[] args) {
    new MyClass().client();
  }
}
```

```
class MyClass {
  { Token_Byte }
  void client() {
    { Init_Byte * L_cur > L_Byte * Token_Byte }
      { cache -> data * ... }
    Byte zero = Byte.cache[0];
      { cache -> data * ... }
    { Token_Byte }
  }
}
```

In the paper

- Duplicable module invariants:
  - Never broken after they are established
  - Can be opened without Token_{M}
  - Useful for read-only state

In the paper

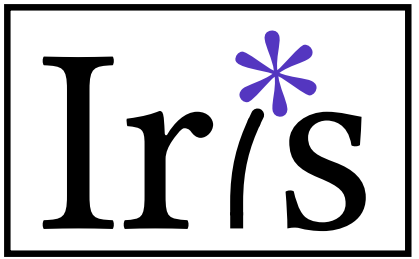
- Duplicable module invariants:
  - Never broken after they are established
  - Can be opened without Token_{M}
  - Useful for read-only state

- Preventing initialization cycles
  - Lead to access of uninitialized data or deadlocks
  - Prevented by initialization levels

```
class A {
  static int a = B.b + 1;
}

class B {
  static int b = A.a + 1;
}
```

Formalization

- Formalized as concurrent separation logic based on Iris



- Proved sound with Rocq

- Formalized as encoding into HeapLang
  - Expressed as concurrency-problem
  - Each initializer is its own thread

Evaluation

- Automated in two verifiers:
  - VerCors for Java (lazy)
  - Gobra for Go (eager)
- Verified small case studies with typical usage patterns

	Go				Java			
Name	LOC	Ann.	MInv.	Time [s]	LOC	Ann.	MInv.	Time [s]
Byte	22	27	8	1.77	18	17	7	1.91
Client*	15	26	9	1.02	13	18	11	1.58
Contexts	35	20	5	1.17	38	21	5	1.34
Counter	24	55	8	1.44	43	59	15	2.03
Fib	14	21	6	0.88	19	18	6	1.41
Fib Conc.	21	29	9	1.37	25	28	9	1.79
Finalizer	47	75	21	33.46	57	59	21	21.35
Names*	42	25	4	2.25	35	24	11	2.26
Singleton*	21	24	2	0.61	24	20	10	4.24
Subclass	-	-	-	-	26	10	9	1.35
Theory*	-	-	-	-	27	23	11	1.69

Evaluation: Part 2

- Used in case study 4.7k lines of real-world Go code base
  - Next-generation internet router implementation
  - Verified in Gobra
  - Integrated with existing verification effort
    - Replaced previous unjustified assumptions

Conclusion

- First modular verification technique for global variable initialization
- Based on Concurrent Separation Logic
- Applies to a wide range of languages
- Abstracts over exact initialization time and semantics

- Proved sound in Iris
- Implemented in VerCors and Gobra