Microsoft

Improving HotSpot Scalar Replacement

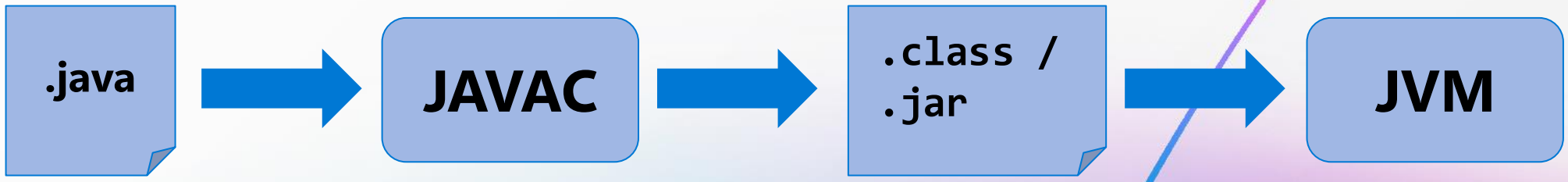
Cesar Soares aka @JohnTortugo
Senior Compiler Engineer

Microsoft Developer Division
Java Engineering Group (JEG)

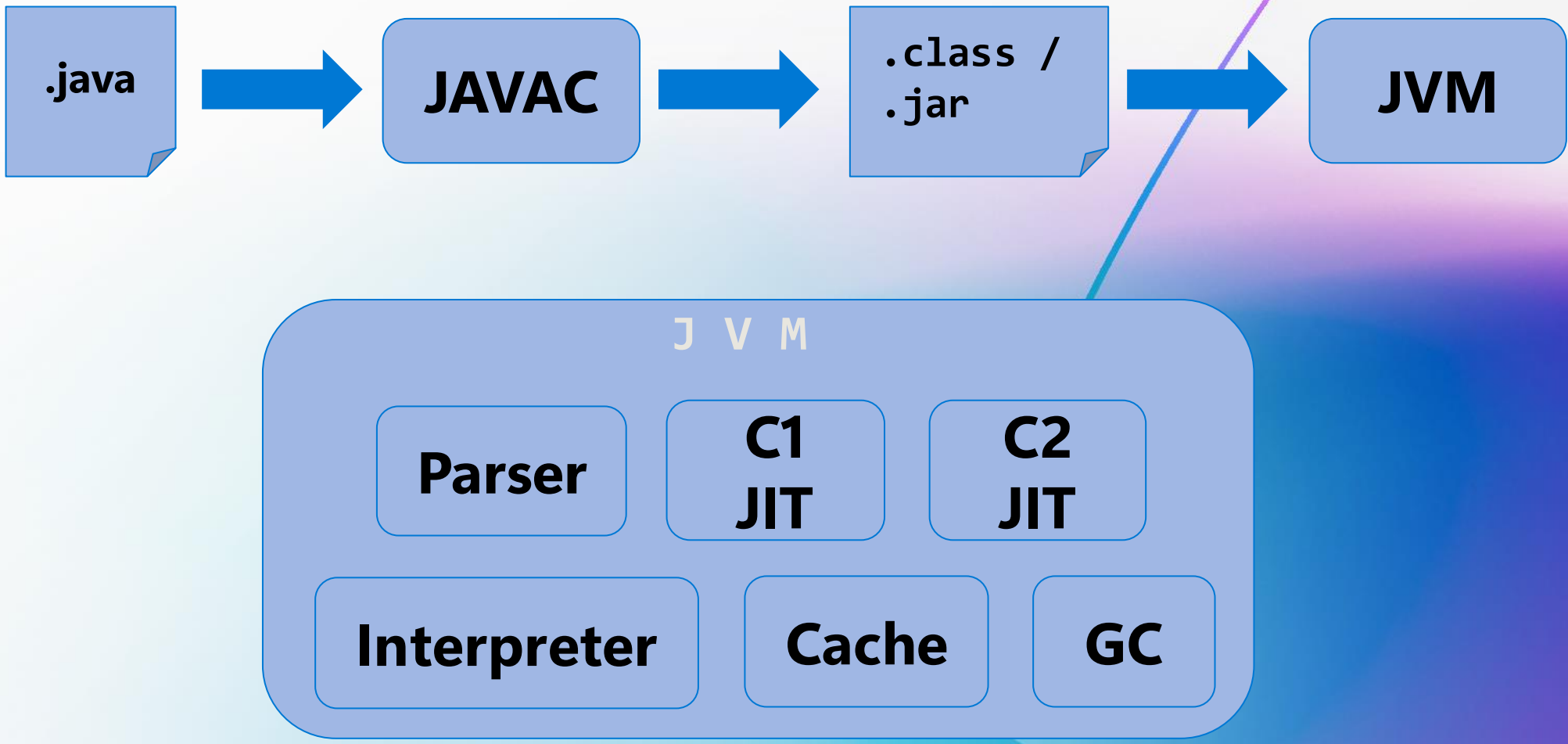
Agenda

1. What is Scalar Replacement?
   Why is it important?
2. How did we improve HotSpot Scalar Replacement?
3. Show me the numbers!
4. Conclusion

What is OpenJDK HotSpot?



What is OpenJDK HotSpot?



What is Scalar Replacement?

- Scalar Replacement is a compiler optimization.
- Built on top of Escape Analysis and Method Inlining.

- It decomposes objects into its constituent fields.
- **The goal:** to remove object allocations.

Why is it important?

Fewer Allocations

Help to reduce number of objects allocated and therefore helps to increases cache locality and reduce time spent in GC.

Less Code

Transforms memory access into register access. Remove need of pointer manipulation to access fields.

Better Code

Simplify the code and make it more amenable to other optimizations.

Running Example

We'll use this class throughout the examples.

The important part is the *Checksum* method.

```
public class Message() {
  String content;

  public Message(String content) {
    this.content = content;
  }

  public int Checksum() {
    int chks = 0;
    for (int i=0; i<content.length(); i++) {
       chks += content.charAt(i);
    }
    return chks;
  }
}
```

The Simple Case

Before Scalar Replacement

```
public int CompositeChecksum(List<String> messages) {
    int checksum = 0;
    for (String msg : messages) {
        Message m = new Message(msg);
        int cs = m.CheckSum();
        checksum += cs;
    }
    return checksum;
}
```

The Simple Case

During Scalar Replacement

```
public int CompositeChecksum(List<String> messages) {
    int checksum = 0;
    for (String msg : messages) {
        <Message's constructor code copied here>
        int cs = <Message's CheckSum() method copied here>
        checksum += cs;
    }
    return checksum;
}
```

The Simple Case

During Scalar Replacement

```
public int CompositeChecksum(List<String> messages) {
    int checksum = 0;
    for (String msg : messages) {
        String content = msg;
        int chks = 0;
        for (int i=0; i<content.length(); i++) chks += content.charAt(i);
        int cs = chks;
        checksum += cs;
    }
    return checksum;
}
```

The Simple Case

After Scalar Replacement

```
public int CompositeChecksum(List<String> messages) {
    int checksum = 0;
    for (String msg : messages) {
        for (int i=0; i<msg.length(); i++)
            checksum += msg.charAt(i);
    }
    return checksum;
}
```

Improving HotSpot Scalar Replacement

A Not So Simple Case

Control Flow Merge

```
public int CompositeChecksum(List<String> messages) {
    int checksum = 0;
    for (String msg : messages) {
        Message m = msg != null ? new Message(msg) : new Message("Clear");
        int cs = m.CheckSum();
        checksum += cs;
    }
    return checksum;
}
```

See other cases here: *https://tinyurl.com/2dwb3z3e*

A Not So Simple Case

Control Flow Merge

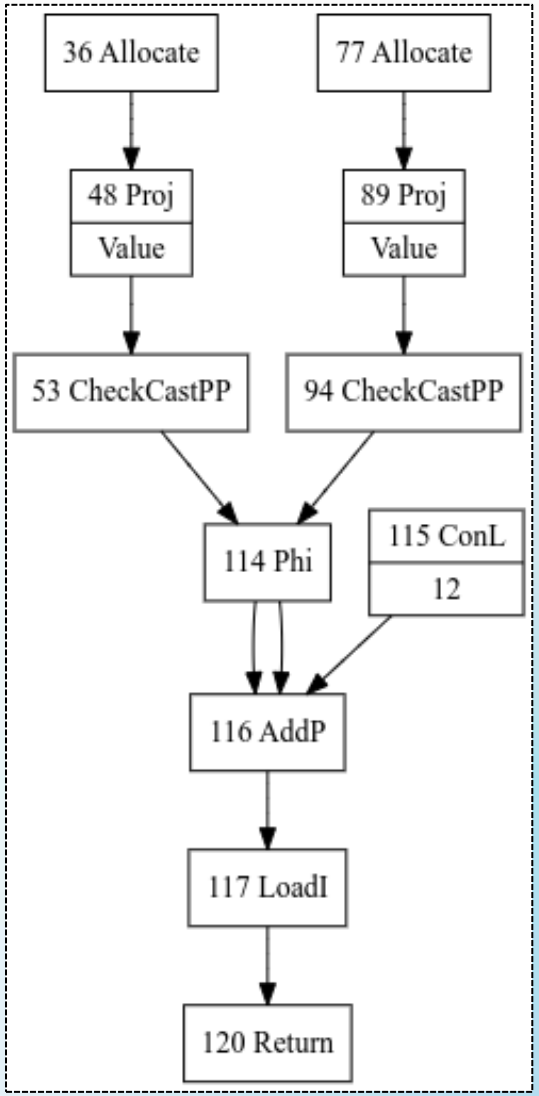
```
public int CompositeChecksum(List<String> messages) {
    int checksum = 0;
    for (String msg : messages) {
        Message m = msg != null ? new Message(msg) : new Message("Clear");
        int cs = m.CheckSum();
        checksum += cs;
    }
    return checksum;
}
```

See other cases here: *https://tinyurl.com/2dwb3z3e*

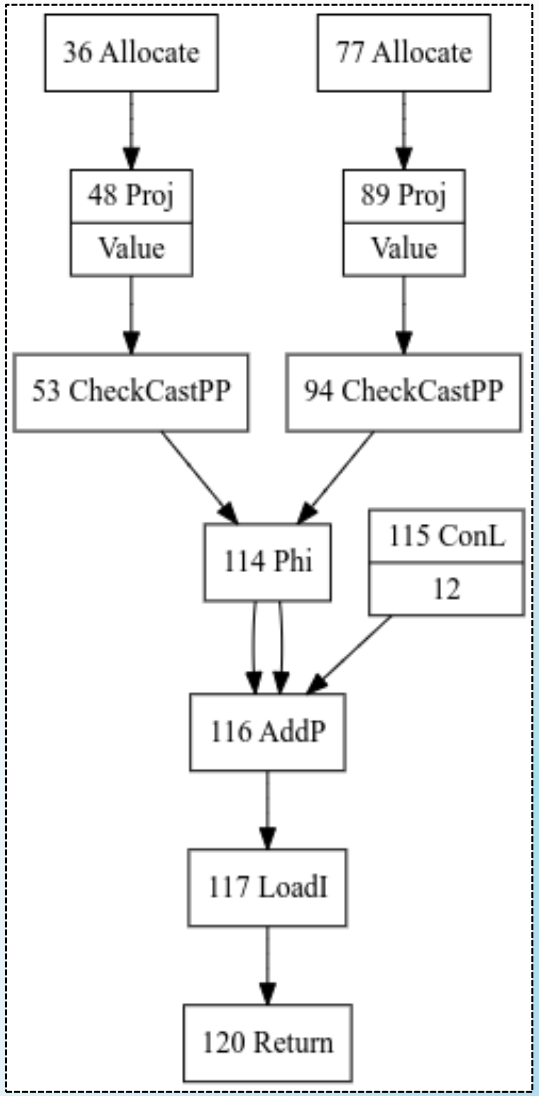
```
public static String whichPayload(String payload) {
    Message m = (payload != null) ?
        new Message(payload) :
        new Message(“Clear”);

    return m.content;
}
```

Let's suppose all we must worry about are field loads.

How C2 Represents this Method

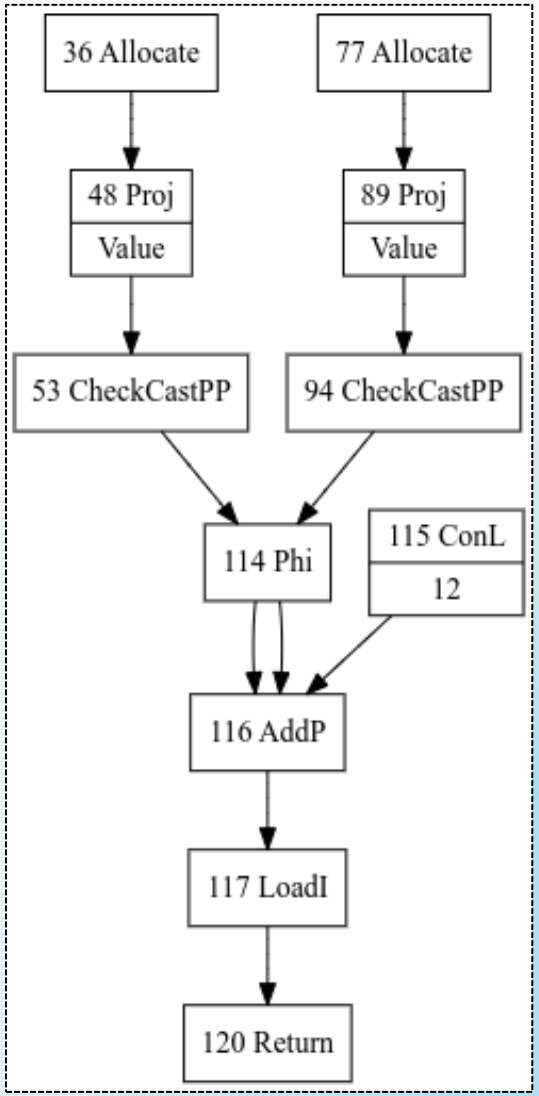


How C2 Represents this Method



Allocate the objects

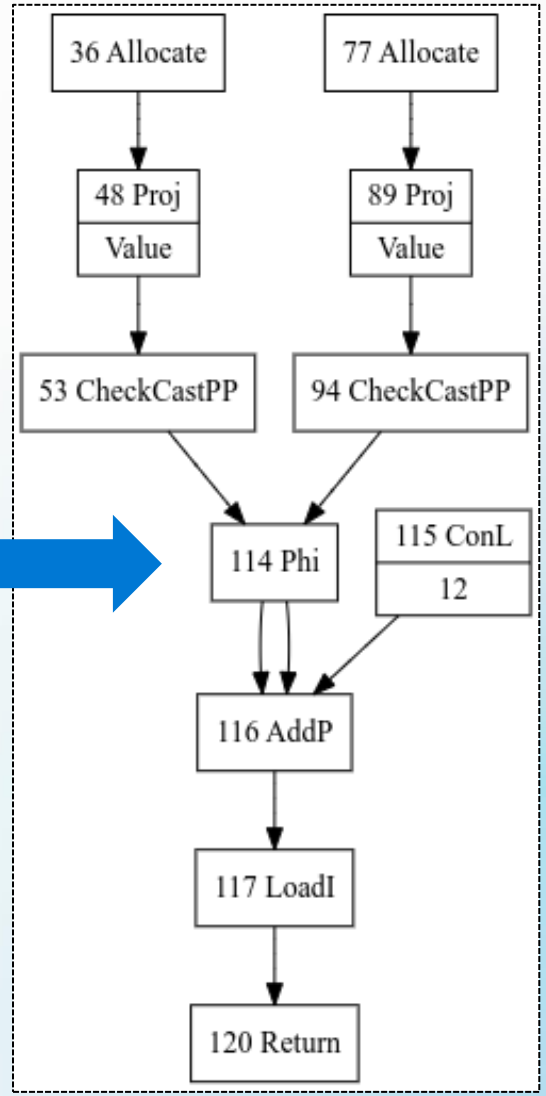
How C2 Represents this Method



Allocate the objects

Cast memory pointer to object type

How C2 Represents this Method

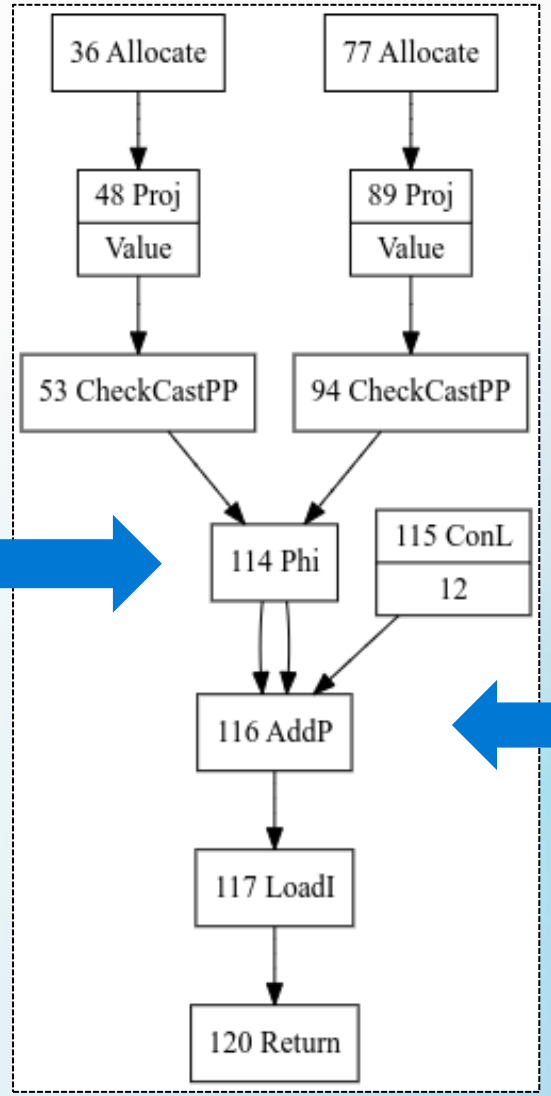


Allocate the objects

Cast memory pointer to object type

Decide which object to return

How C2 Represents this Method



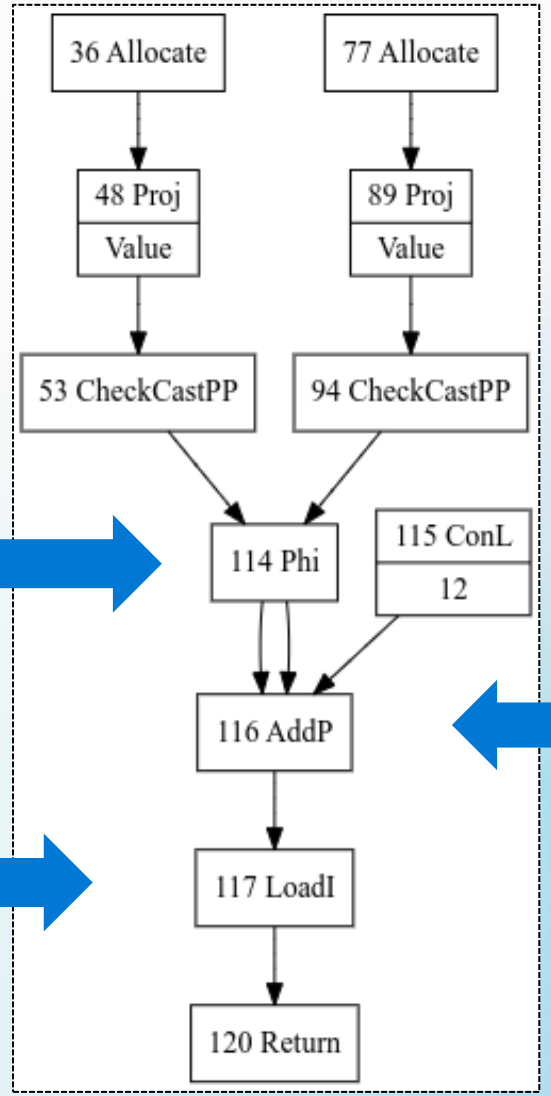
Allocate the objects

Cast memory pointer to object type

Decide which object to return

Compute address of field at offset 12

How C2 Represents this Method



Allocate the objects

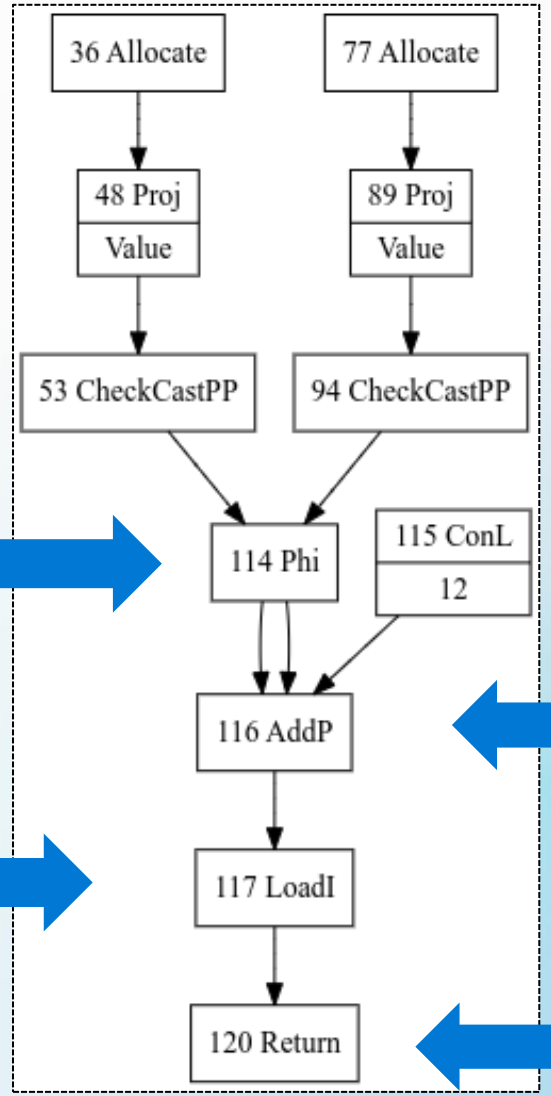
Cast memory pointer to object type

Decide which object to return

Compute address of field at offset 12

Load field value

How C2 Represents this Method



Allocate the objects

Cast memory pointer to object type

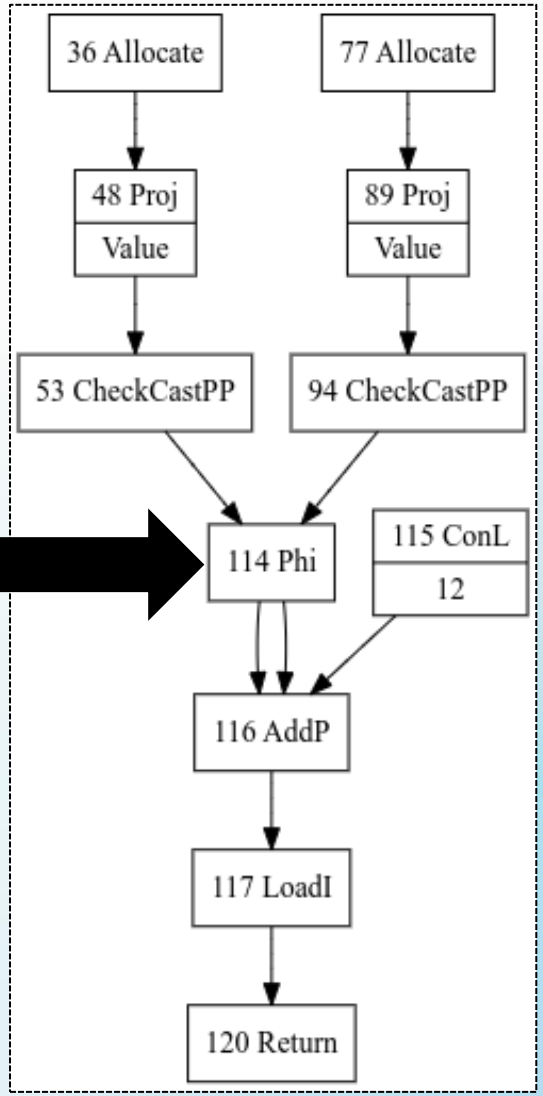
Decide which object to return

Compute address of field at offset 12

Load field value

Return the value

How C2 Represents this Method

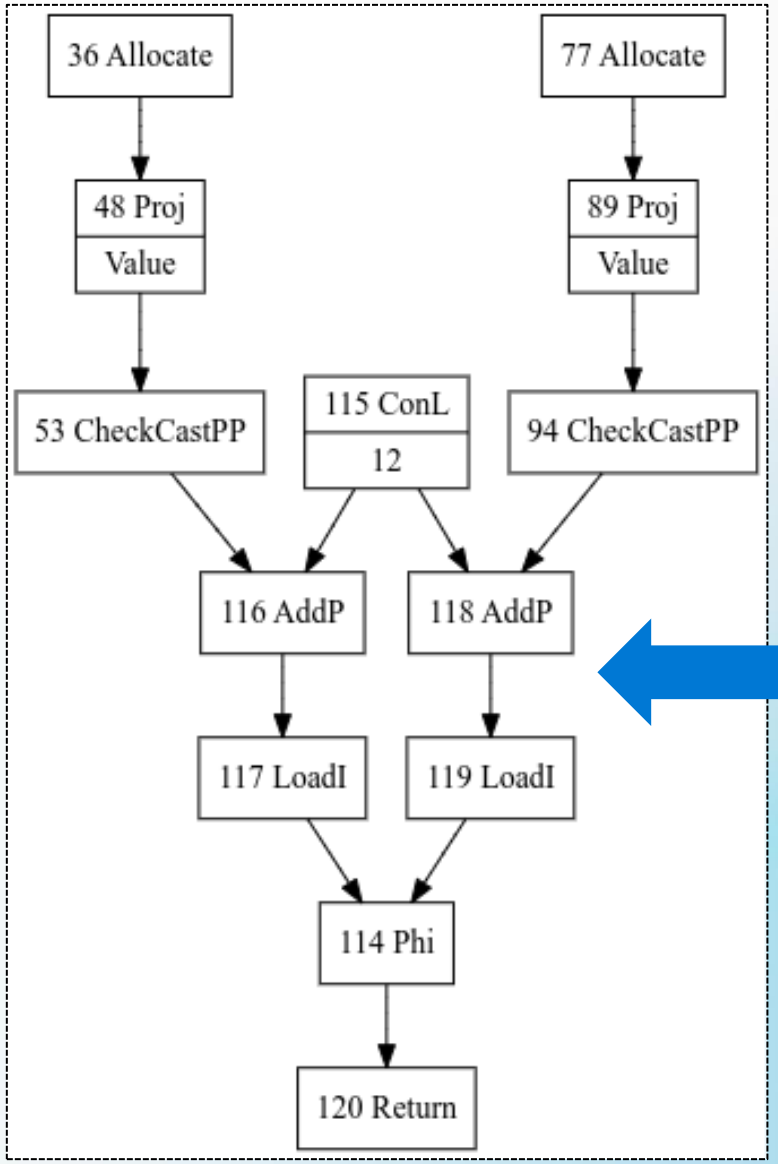


Decide which object to return

C2 does *NOT* scalar replace the objects if the field load is *AFTER* the decision of which object to load the field from.

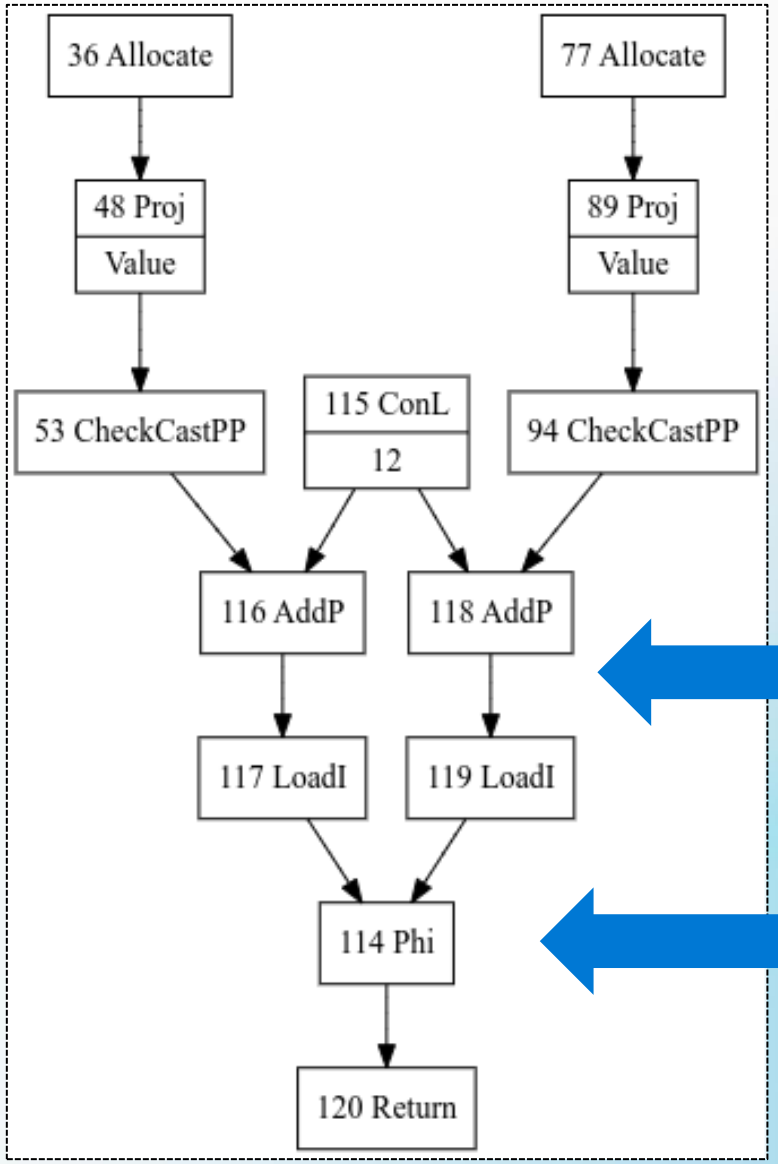
How did we solve it

How did we solve it



Represent the field load first.

How did we solve it



Represent the field load first.

Decide which load should be used instead of which object.

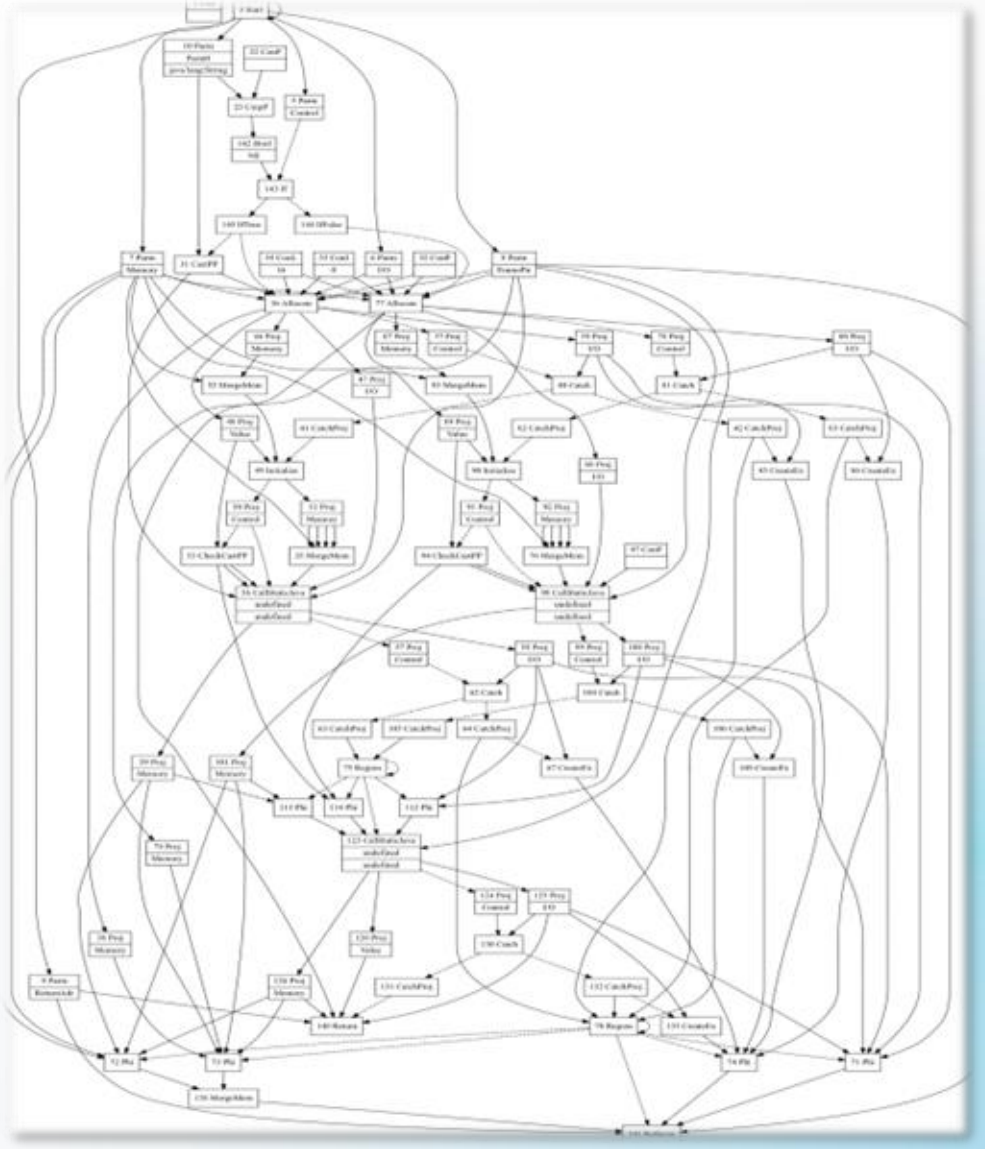
```
public static String whichPayload(String payload) {
    Message content = (payload != null) ?
        new Message(payload).content :
        new Message("Clear").content;

    return content;
}
```

The conditional is now used to decide which loaded value to use, instead of which object to load from.

```
public static String whichPayload(String payload) {
    return (payload != null) ? payload : “Clear”;
}
```

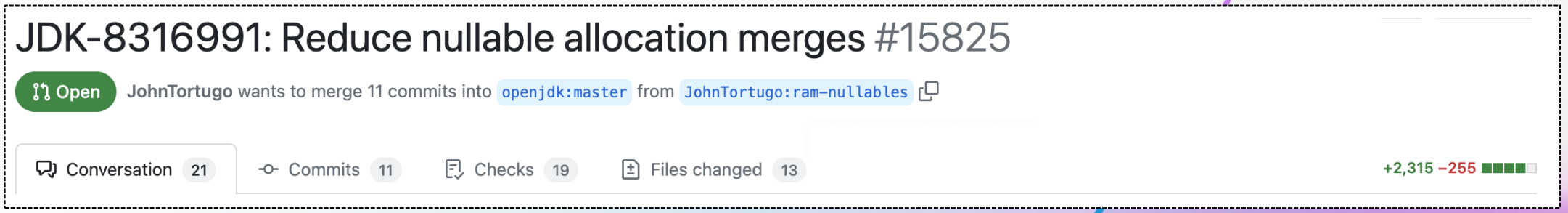
After both objects are scalar replaced.

How C2 Represents this Method



- Thousands of nodes.
- Memory graph updates.
- Cooperate with other optimizations.
- Handle a variety object use cases.
- Handle deoptimizations.
- Don't break existing code.

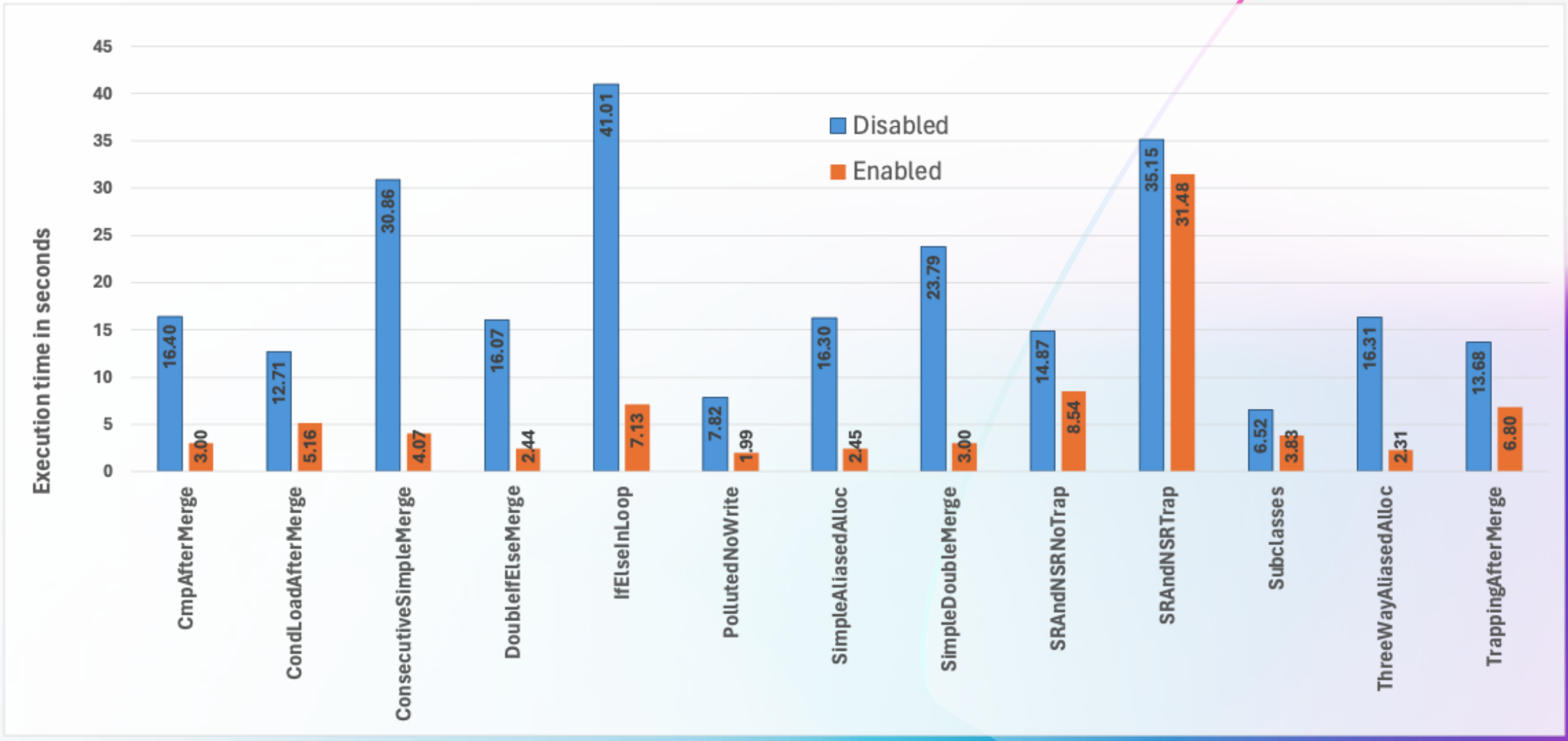
Proposed changes so far



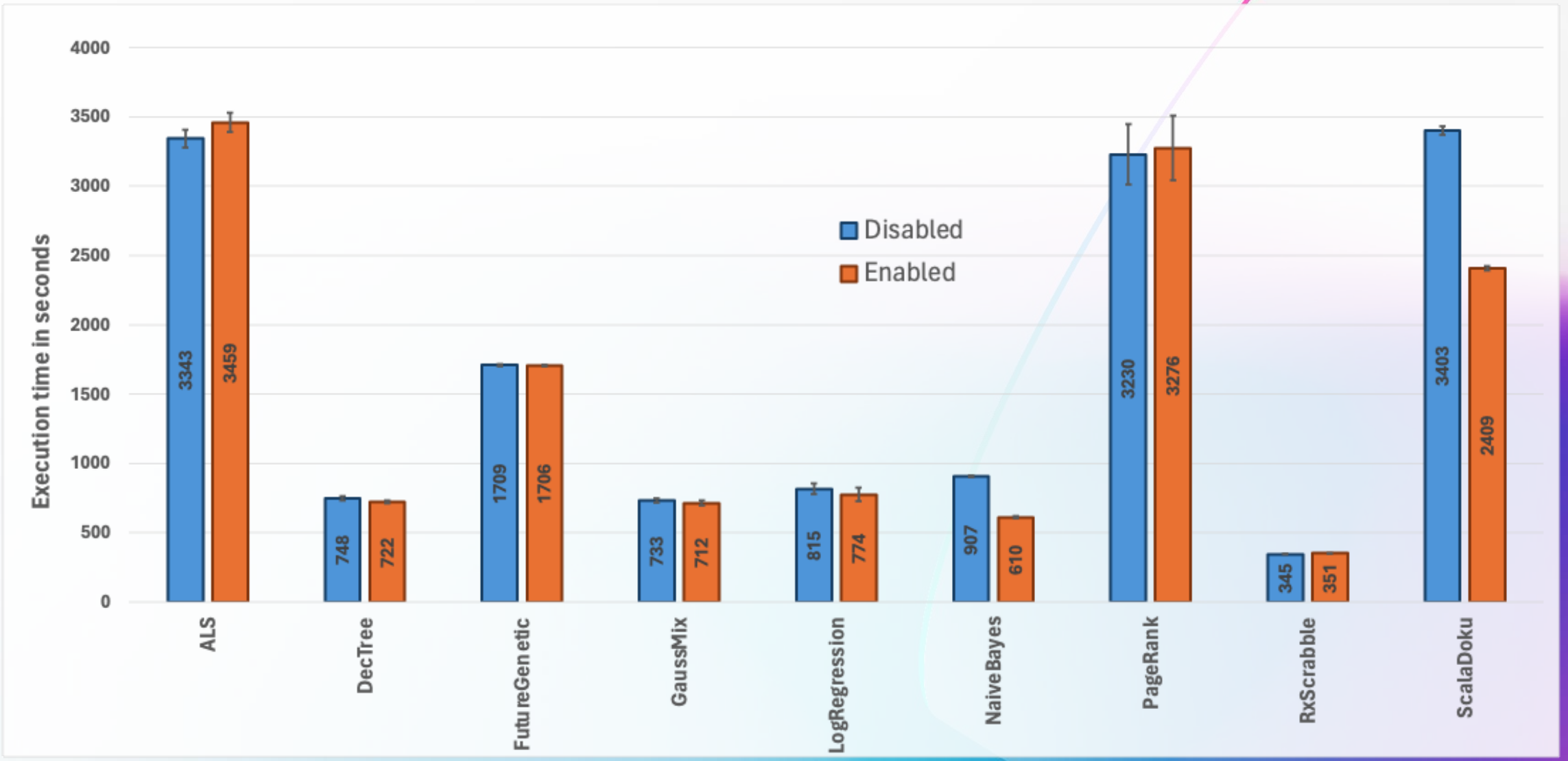
https://github.com/openjdk/jdk/pull/15825

https://github.com/openjdk/jdk/pull/12829

Execution Time of Synthetic Benchmarks



Execution Time of Renaissance Benchmarks



Microsoft

Conclusions

- C2 wasn't scalar replacing some object allocation merges.
- Identifying a solution was "easy enough".
- Implementing the solution was challenging.
- Changes resulted in noticeable performance improvements in several synthetic and real-world benchmarks.

Microsoft

Thank you!

... and feel free to reach out:

@JohnTortugo
disoares@microsoft.com