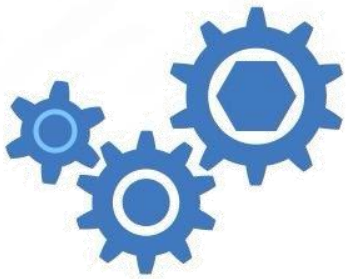
On Control Flow Equivalence

Its Utility in Program Optimization

Dr. Andrew Craik
Advisory Software Developer
IBM Runtime Technologies
1-Nov-2016



IBM Runtime Technologies

IBM

Control Flow Equivalence

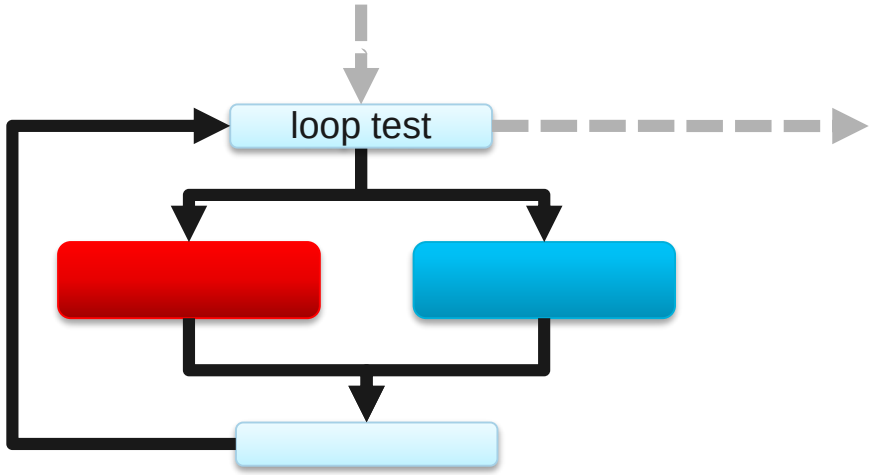
- A statement about two points in a program
- Two points are control-flow equivalent iff the program produces the same observable results when run with the same state from either of the two program points
  Note: internal intermediate states may differ
- Powerful tool for program optimization

Sources of Control Flow Equivalence

- Can occur in a program given to a compiler
  - often a result of hand optimization
- More often it is generated by program optimization
  - Loop Unrolling
  - Guarded Call Inlining
  - Tail Duplication
  - …

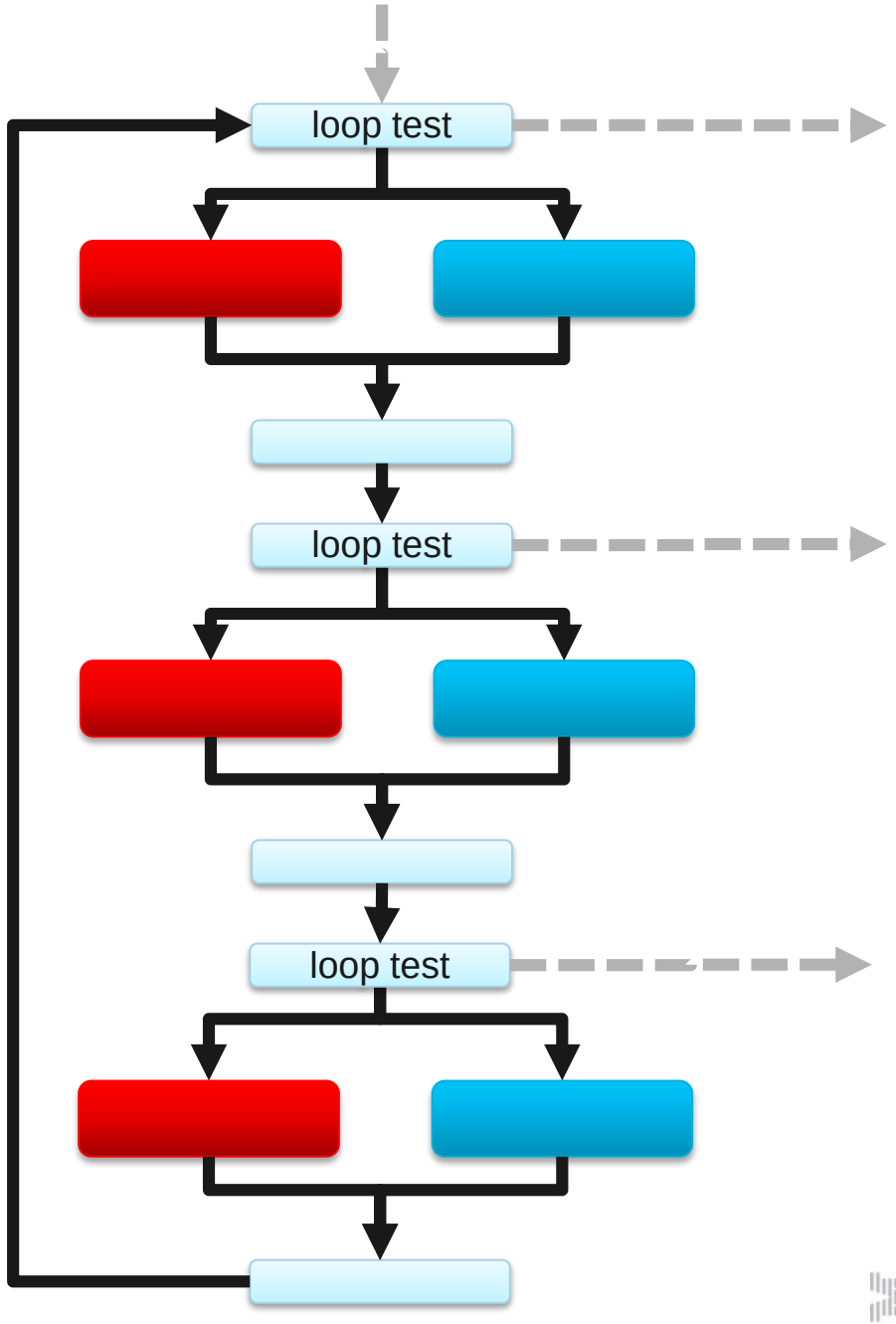
CFE – Loop Unrolling

- Loop Unrolling – replicate loop body to reduce backward branches
- Replicated bodies are connected entry-to-exit with mid-loop exits
- Replicated bodies generate control-flow equivalent points – program execution behavior is the same regardless of the body run



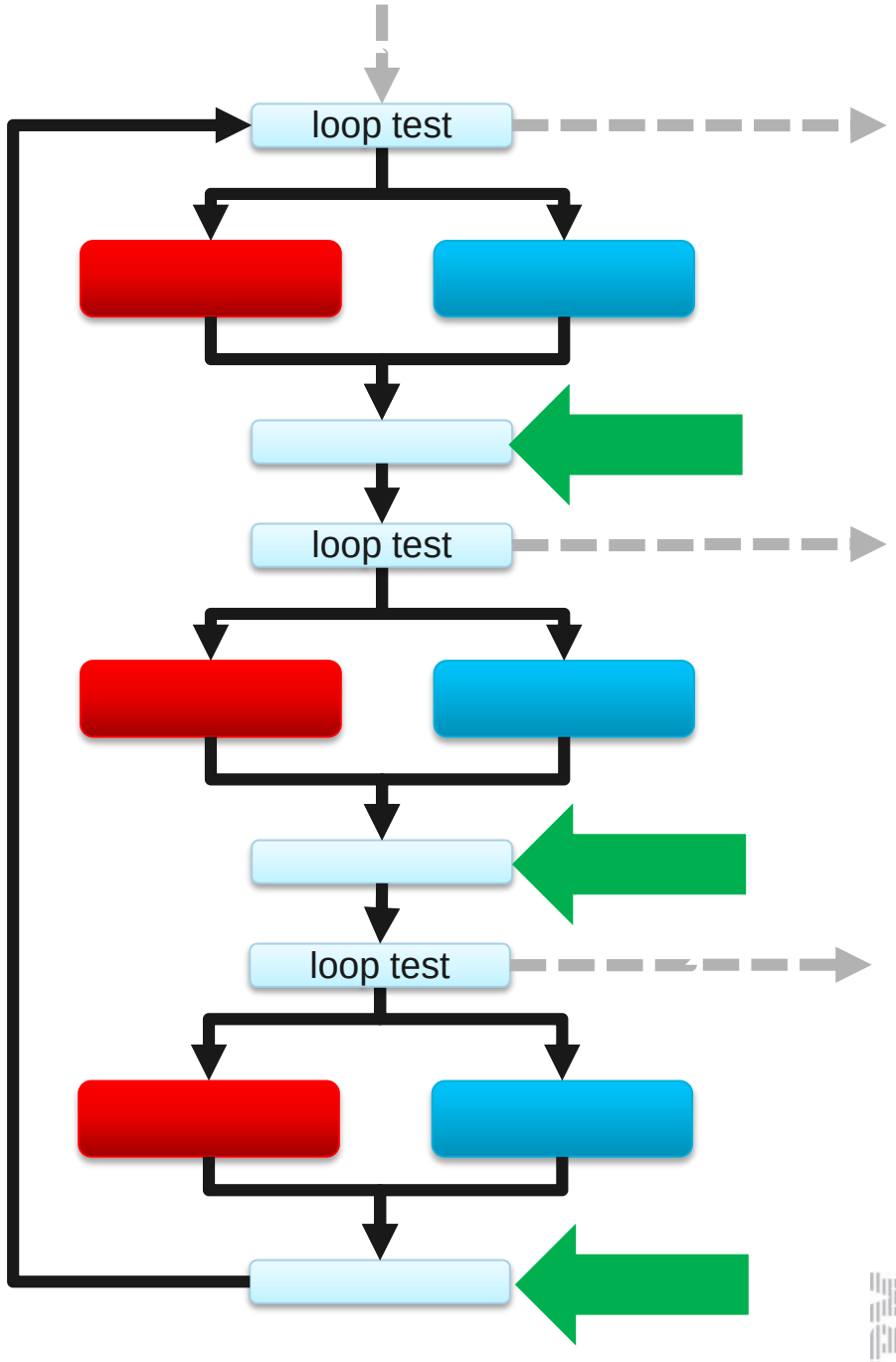
CFE – Loop Unrolling

- Loop Unrolling – replicate loop body to reduce backward branches
- Replicated bodies are connected entry-to-exit with mid-loop exits
- Replicated bodies generate control-flow equivalent points – program execution behavior is the same regardless of the body run



CFE – Loop Unrolling

- Loop Unrolling – replicate loop body to reduce backward branches
- Replicated bodies are connected entry-to-exit with mid-loop exits
- Replicated bodies generate control-flow equivalent points – program execution behavior is the same regardless of the body run

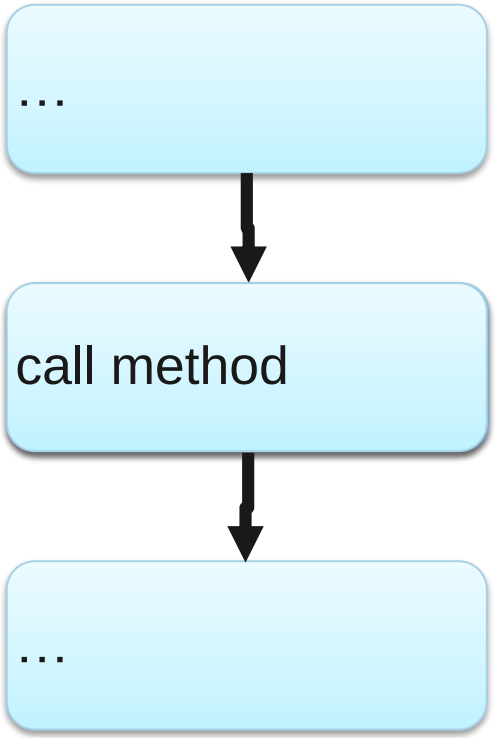


CFE – Guarded Call Inlining

- Calls complicate program optimization
  - Unknown side-effects
  - Limit optimization scope
- Inlining – substitute method implementation for call
- In presence of virtual method resolution precise method may not be known statically
- Inline speculatively and test the body is the right one before running it otherwise call the method as usual

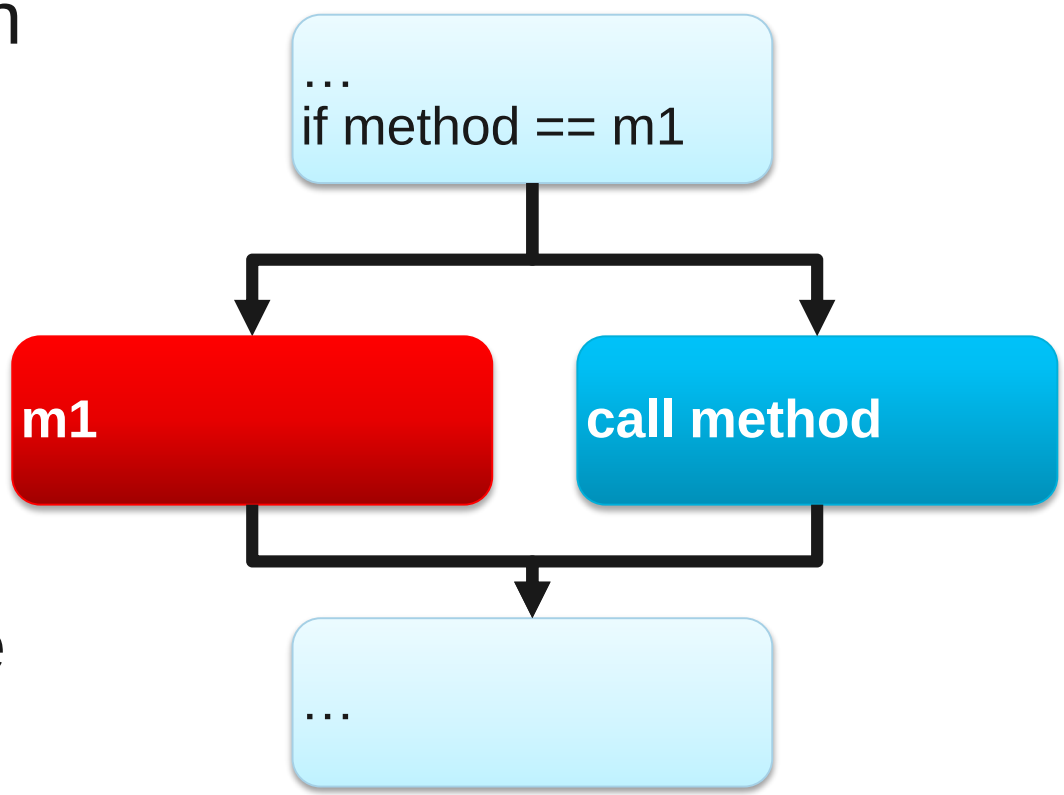
CFE – Guarded Call Inlining

- Calls complicate program optimization
  - Unknown side-effects
  - Limit optimization scope
- Inlining – substitute method implementation for call
- In presence of virtual method resolution precise method may not be known statically
- Inline speculatively and test the body is the right one before running it otherwise call the method as usual



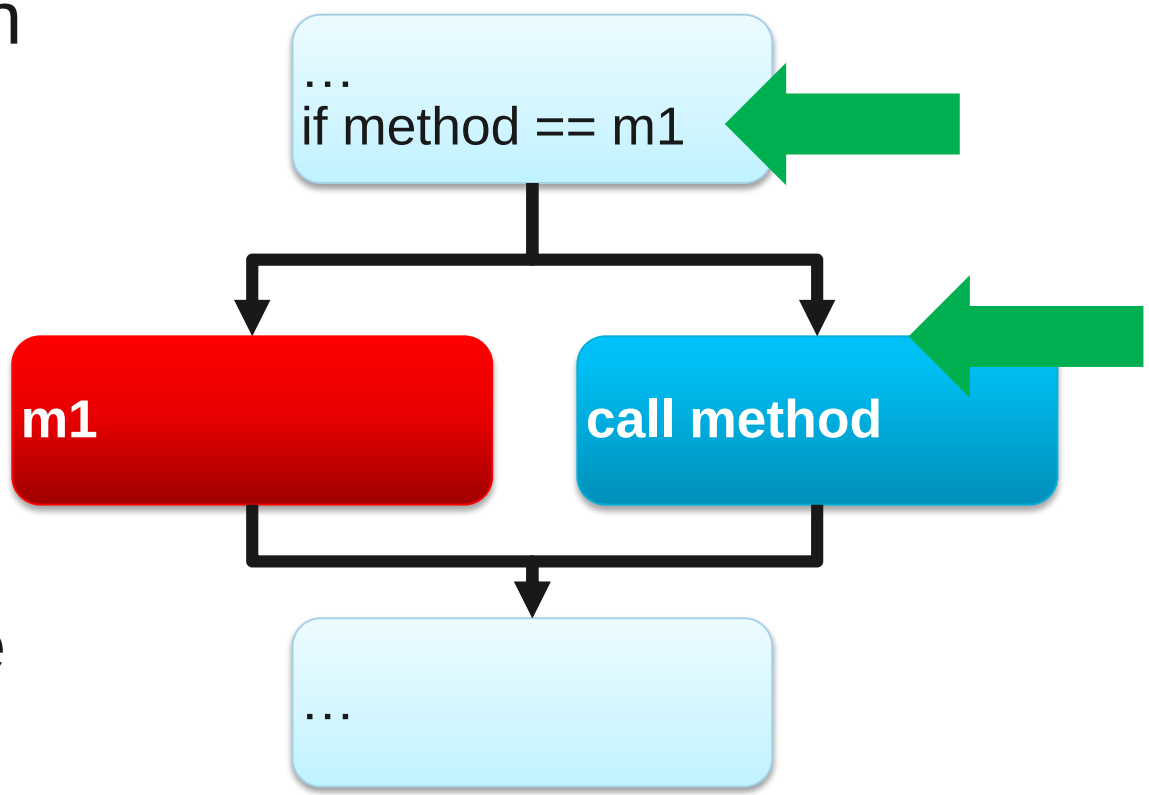
CFE – Guarded Call Inlining

- Calls complicate program optimization
  - Unknown side-effects
  - Limit optimization scope
- Inlining – substitute method implementation for call
- In presence of virtual method resolution precise method may not be known statically
- Inline speculatively and test the body is the right one before running it otherwise call the method as usual



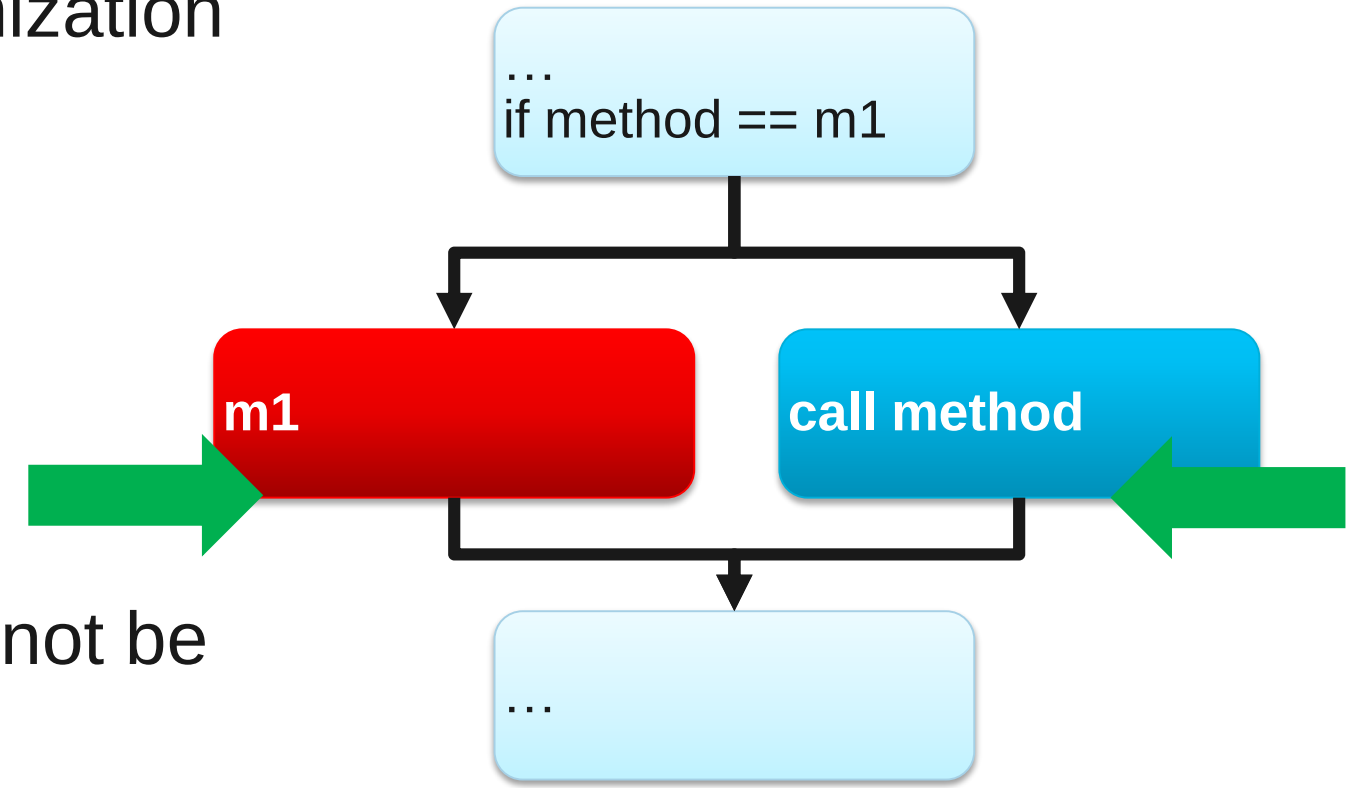
CFE – Guarded Call Inlining

- Calls complicate program optimization
  - Unknown side-effects
  - Limit optimization scope
- Inlining – substitute method implementation for call
- In presence of virtual method resolution precise method may not be known statically
- Inline speculatively and test the body is the right one before running it otherwise call the method as usual



CFE – Guarded Call Inlining

- Calls complicate program optimization
  - Unknown side-effects
  - Limit optimization scope
- Inlining – substitute method implementation for call
- In presence of virtual method resolution precise method may not be known statically
- Inline speculatively and test the body is the right one before running it otherwise call the method as usual



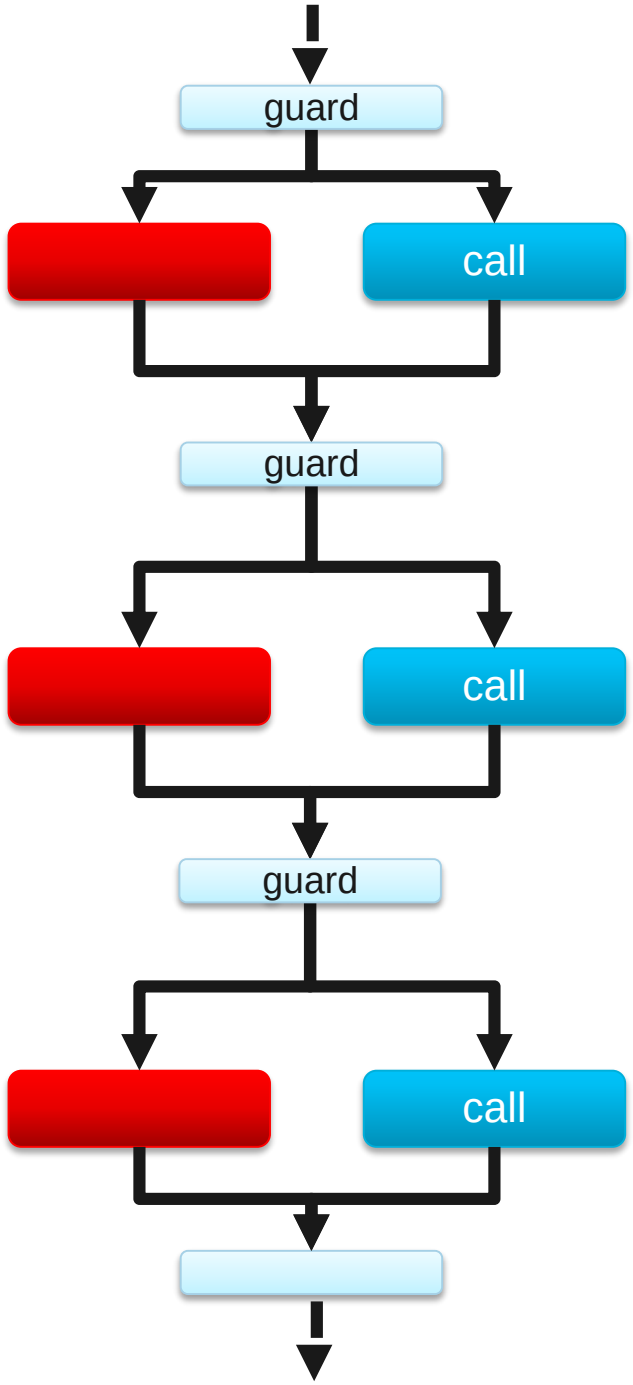
Exploiting Control Flow Equivalence

Utility of Control Flow Equivalence

- Compiler optimizations, especially in a JIT, are often speculative
- Speculative optimizations require a “safe” fallback execution path – a path which is guaranteed to be correct though with lower performance
- Safe fallback paths complicate optimization
  - Disrupt “natural” shape of control flow structures
  - Frequently contain calls (i.e. unknown code)
  - Increase memory footprint of compiled code
- Idea: utilize control flow equivalence to reduce
  - Number of fallback paths
  - Number of control-flow branches & merges for these paths

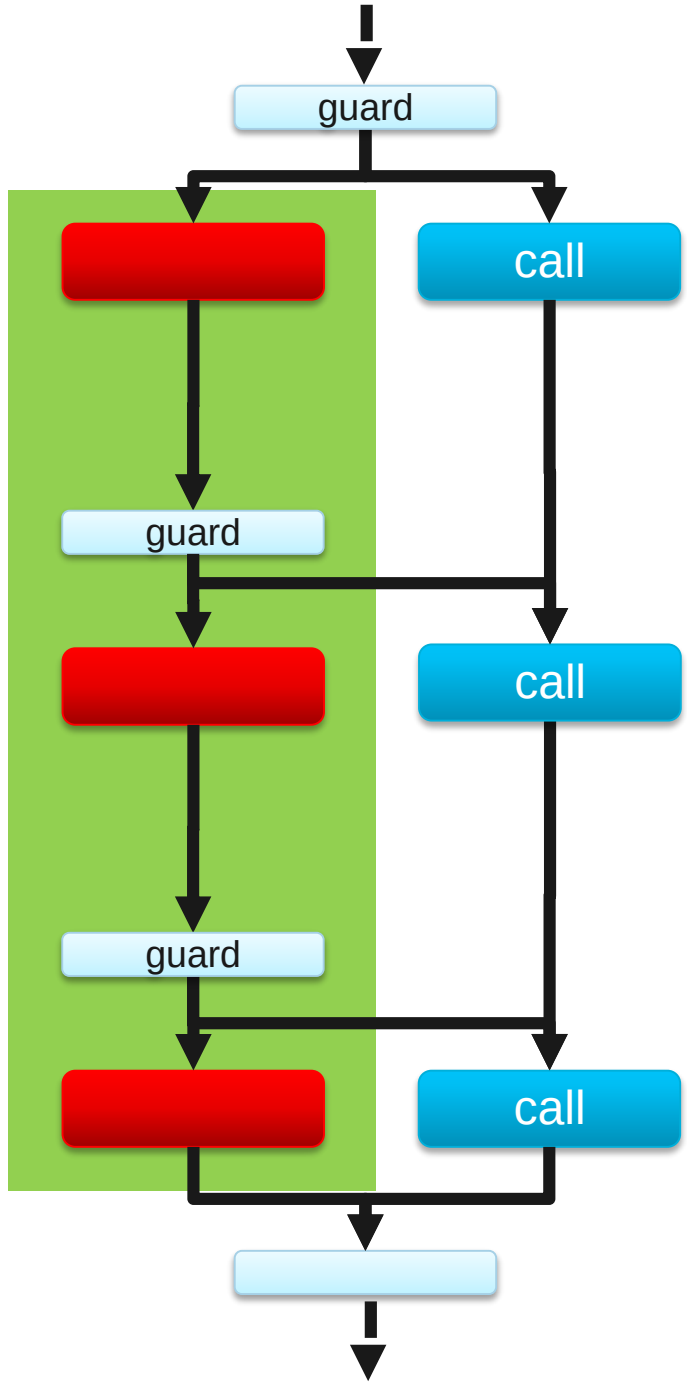
Virtual Guard Tail Splitting

- Consider a program with a sequence of virtual method calls
- Guarded inlining will produce a sequence of control-flow diamonds
- Control-flow merges are expensive – disrupt other optimizations
- Exploit control flow equivalence to reduce merges – cold sides are control flow equivalent with hot sides



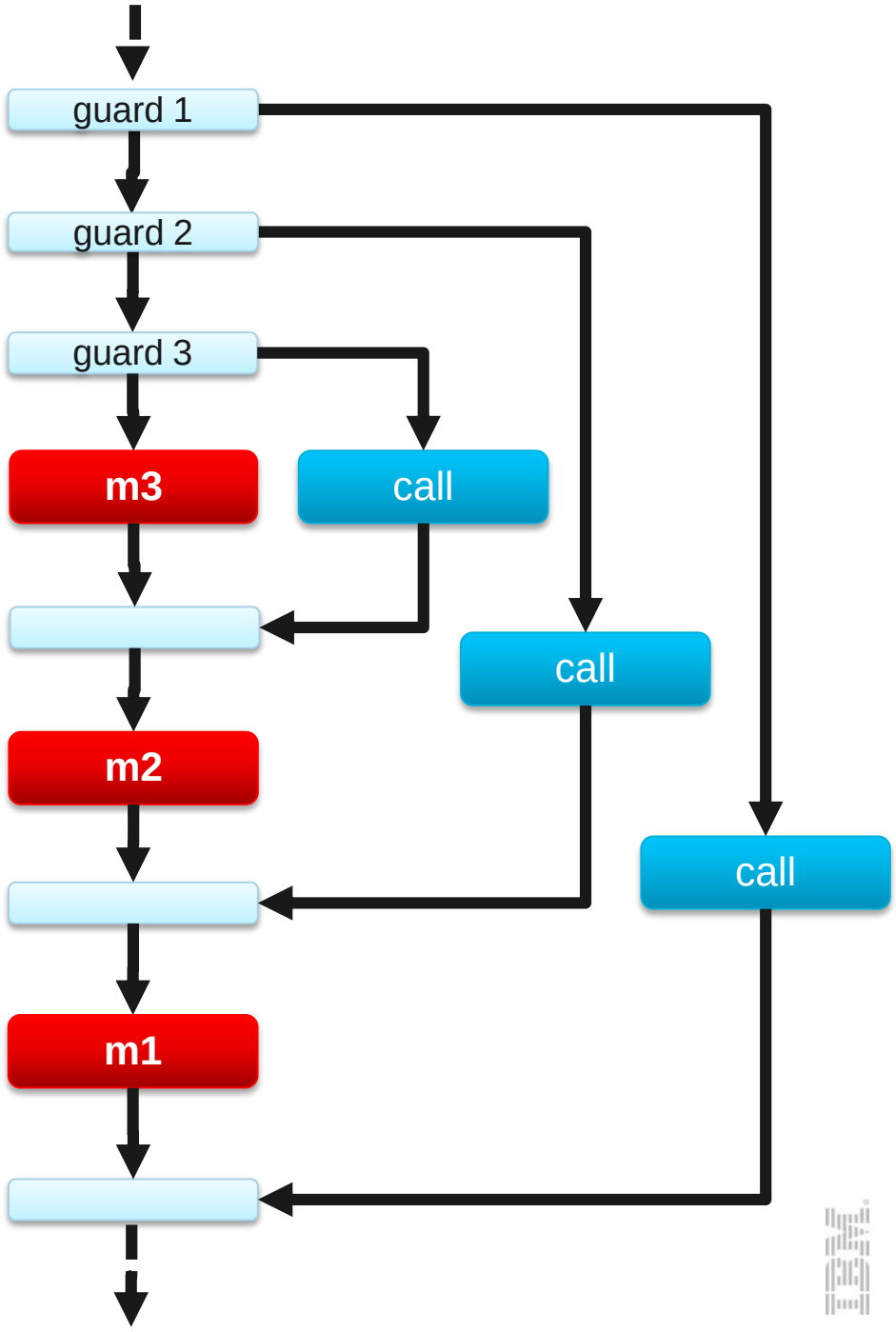
Virtual Guard Tail Splitting

- Consider a program with a sequence of virtual method calls
- Guarded inlining will produce a sequence of control-flow diamonds
- Control-flow merges are expensive – disrupt other optimizations
- Exploit control flow equivalence to reduce merges – cold calls are control flow equivalent points to guarded inline bodies



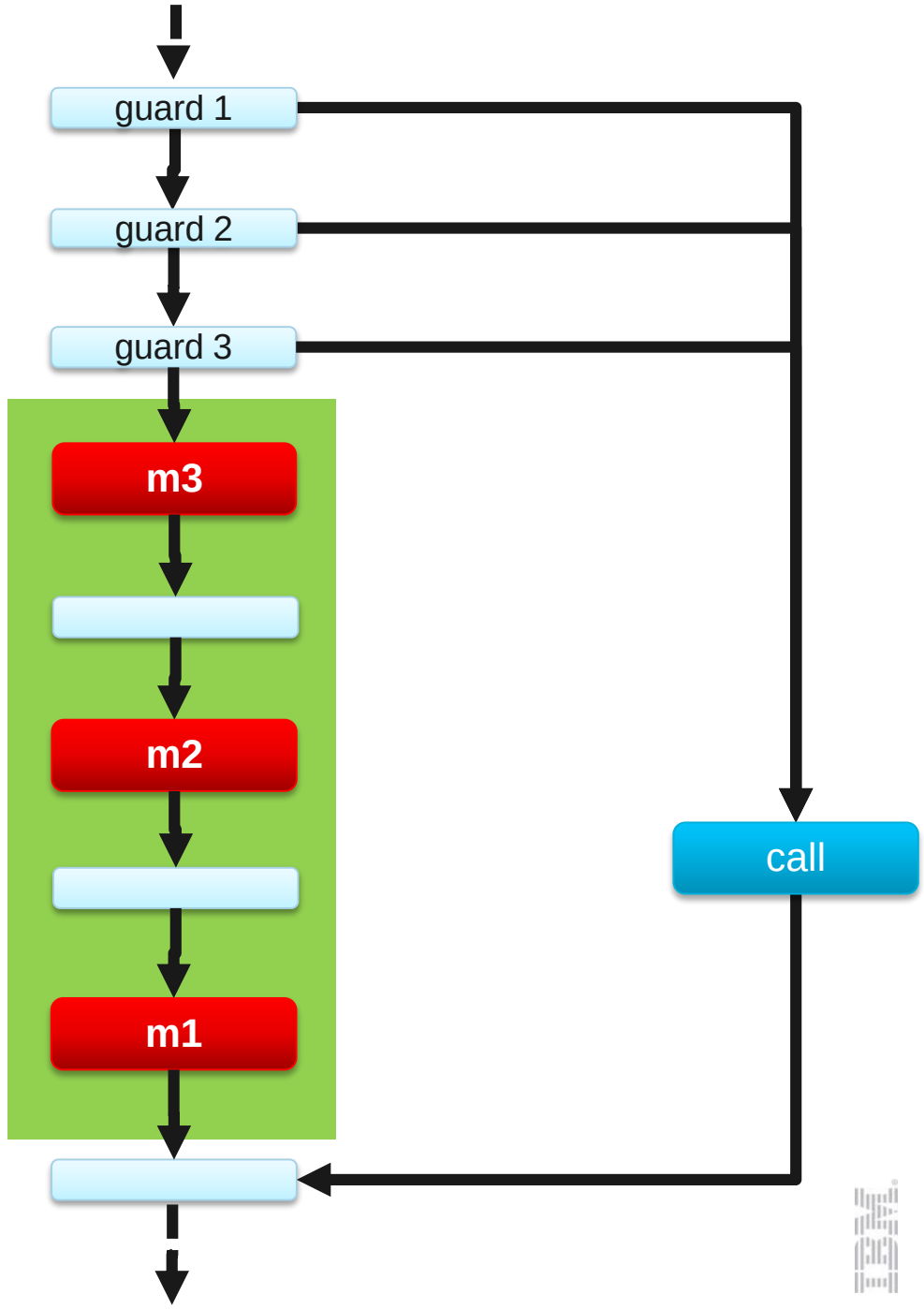
Virtual Guard Head Merging

- Immediately adjacent guards – consider constructors which immediately call super()
- Optimization disrupted by repeated merges from ‘cold’ fallback call paths
- Control-flow equivalence: cannot tell difference between the start of all 3 cold blocks
- Redirect all three guards to the outermost call



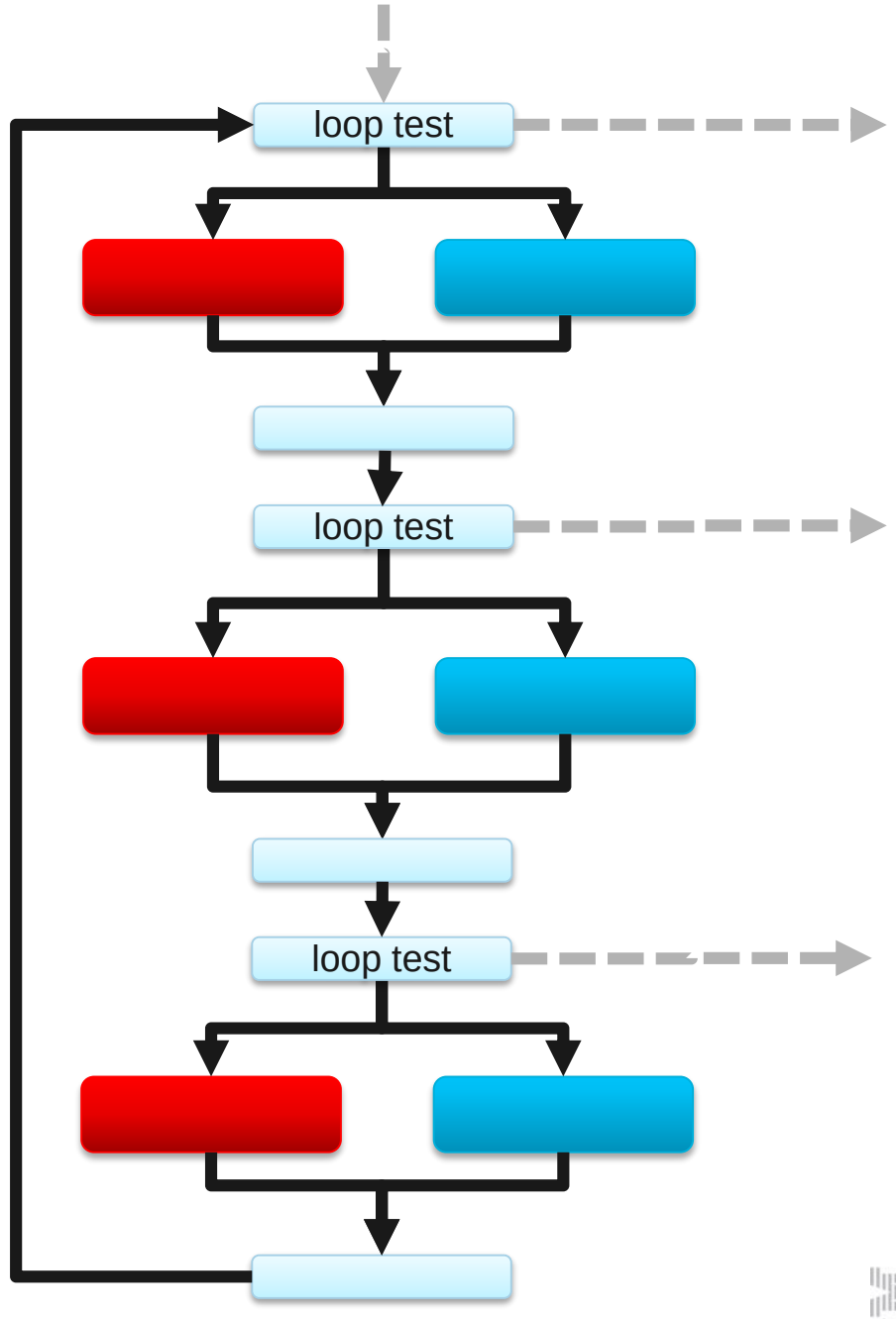
Virtual Guard Head Merging

- Immediately adjacent guards – consider constructors which immediately call super()
- Optimization disrupted by repeated merges from ‘cold’ fallback call paths
- Control-flow equivalence: cannot tell difference between the start of all 3 cold blocks
- Redirect all three guards to the outermost call



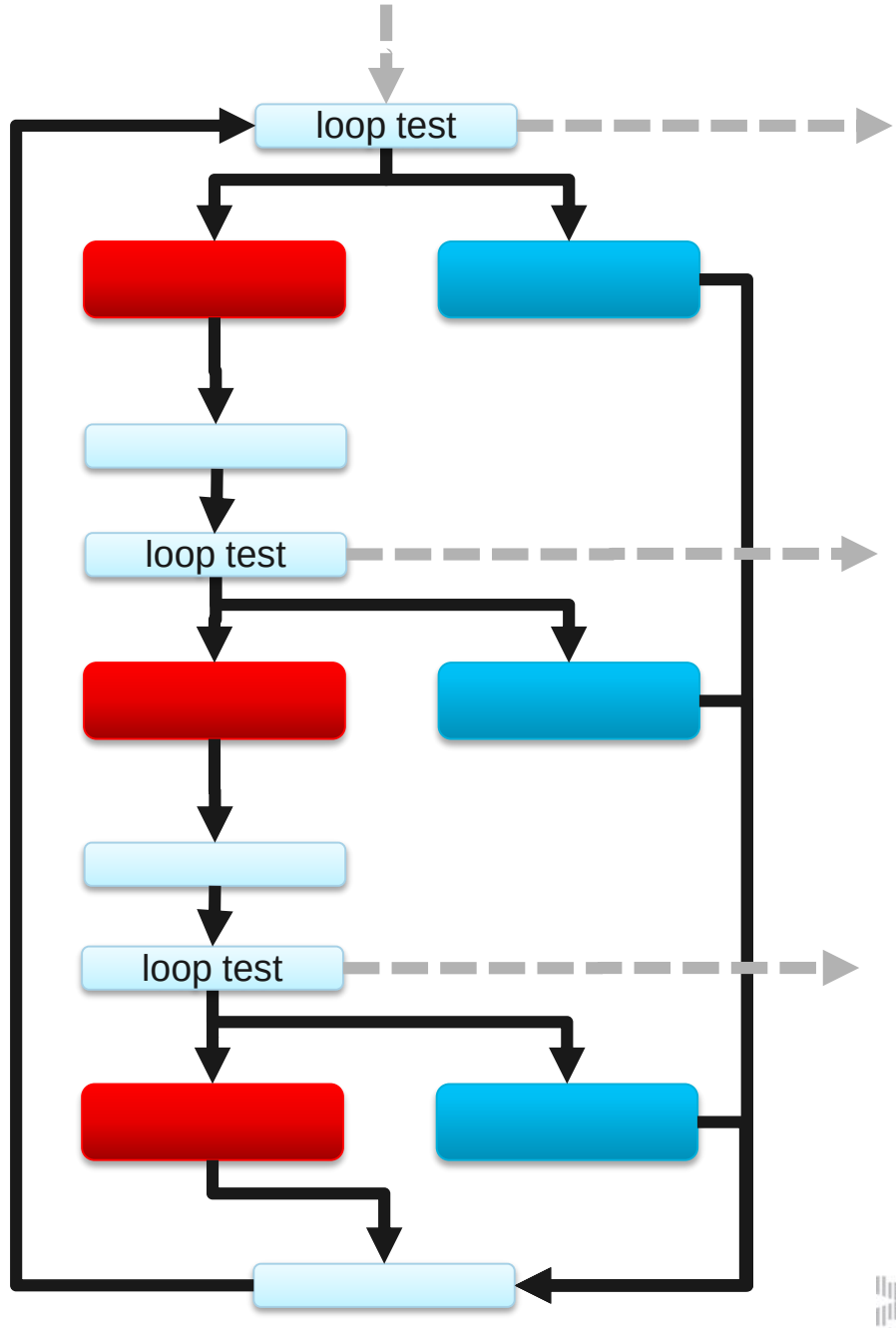
Partial Edge Redirection

- Unrolled iteration connections merge hot & cold execution paths
- Insight: the loop header and the start of an unrolled iteration are control-flow equivalent points
- Idea: redirect cold paths to loop header ⇒ more scope for optimizing the hot path(s)



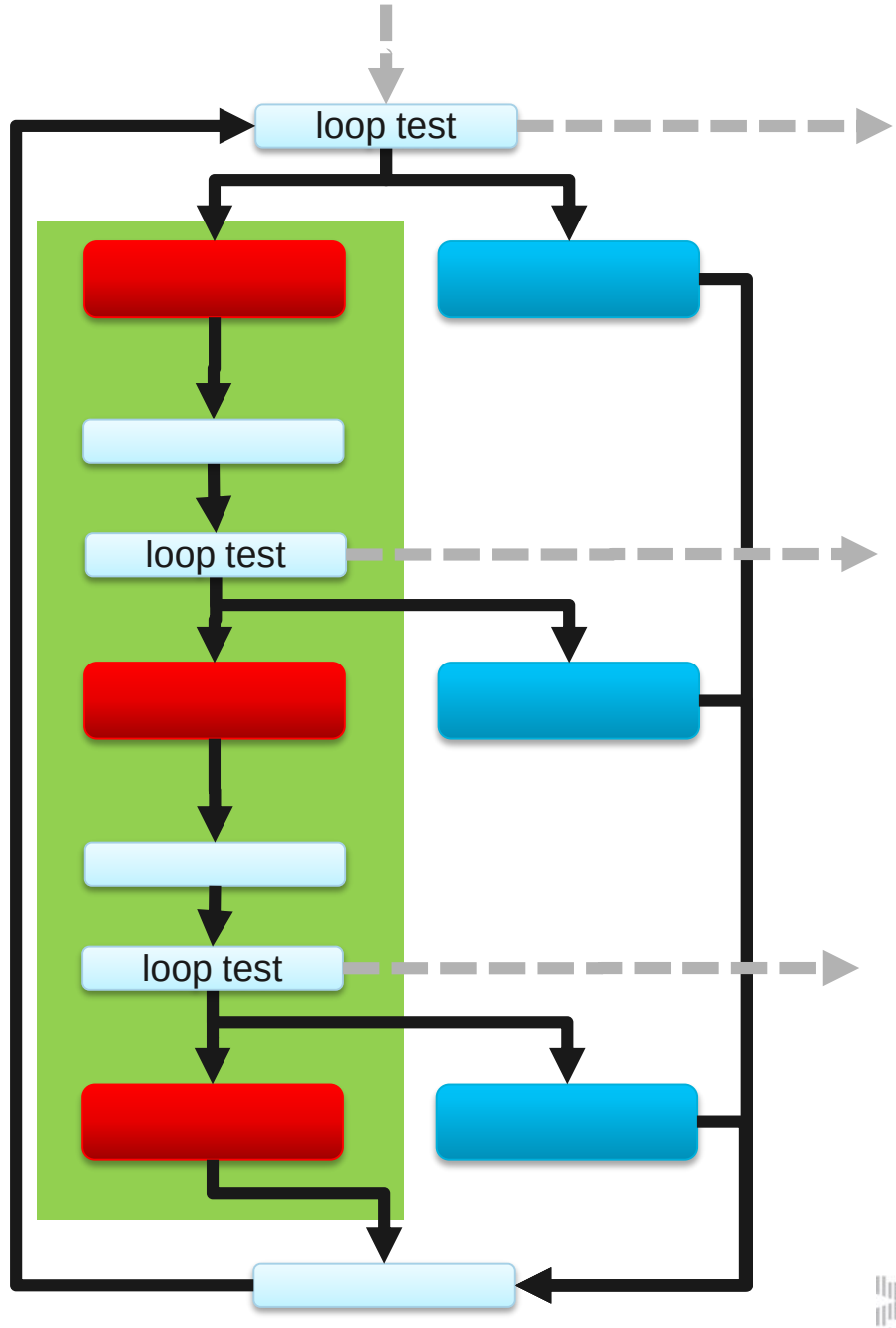
Partial Edge Redirection

- Unrolled iteration connections merge hot & cold execution paths
- Insight: the loop header and the start of an unrolled iteration are control-flow equivalent points
- Idea: redirect cold paths to loop header ⇒ more scope for optimizing the hot path(s)



Partial Edge Redirection

- Unrolled iteration connections merge hot & cold execution paths
- Insight: the loop header and the start of an unrolled iteration are control-flow equivalent points
- Idea: redirect cold paths to loop header ⇒ more scope for optimizing the hot path(s)
- Creates an extended basic block



Conclusion

- Control Flow Equivalence is a by-product of many common optimizations:
  - Guarded Inlining
  - Loop Unrolling
  - …
- Control Flow Equivalence is, itself, useful for optimization
  - Virtual Guard Head Merging
  - Virtual Guard Tail Splitting
  - Partial Edge Redirection
  - …
- Control Flow Equivalence can provide “free” fallback paths

Thank You

Q&A

ajcraik@ca.ibm.com