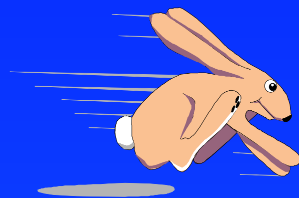
# Eliminating Exception Constraints of Java Programs for IA-64



<http://www.trl.ibm.com/projects/jit/index_e.htm>

Kazuaki Ishizaki, Tatsushi Inagaki,
Hideaki Komatsu,Toshio Nakatani

IBM Research,
Tokyo Research Laboratory

# Goal of the Paper

- Motivation
  - Enable to perform code motion to exploit *instruction level parallelism* (ILP) of IA-64 for Java
  - Enable to perform only beneficial speculative code motion
- Our approach "*exception speculation*" using speculative code motion
  - ► Perform exception speculation on *directed acyclic graph* (DAG)
- Experimental results
- Summary

# A Running Example

- **Java program and bytecode**

**Java program**

```
int foo(int a[], int i) {
  return a[i] + 1;
}
```

**Bytecode**

```
iaload
iconst_1
iadd
ireturn
```

**PEI (Potentially Excepting Instruction) may throw a Java exception.**

# Intermediate Representation

- **Java language introduces many exception checks**

**Java program**

```
int foo(int a[], int i) {
  return a[i] + 1;
}
```

**Bytecode**

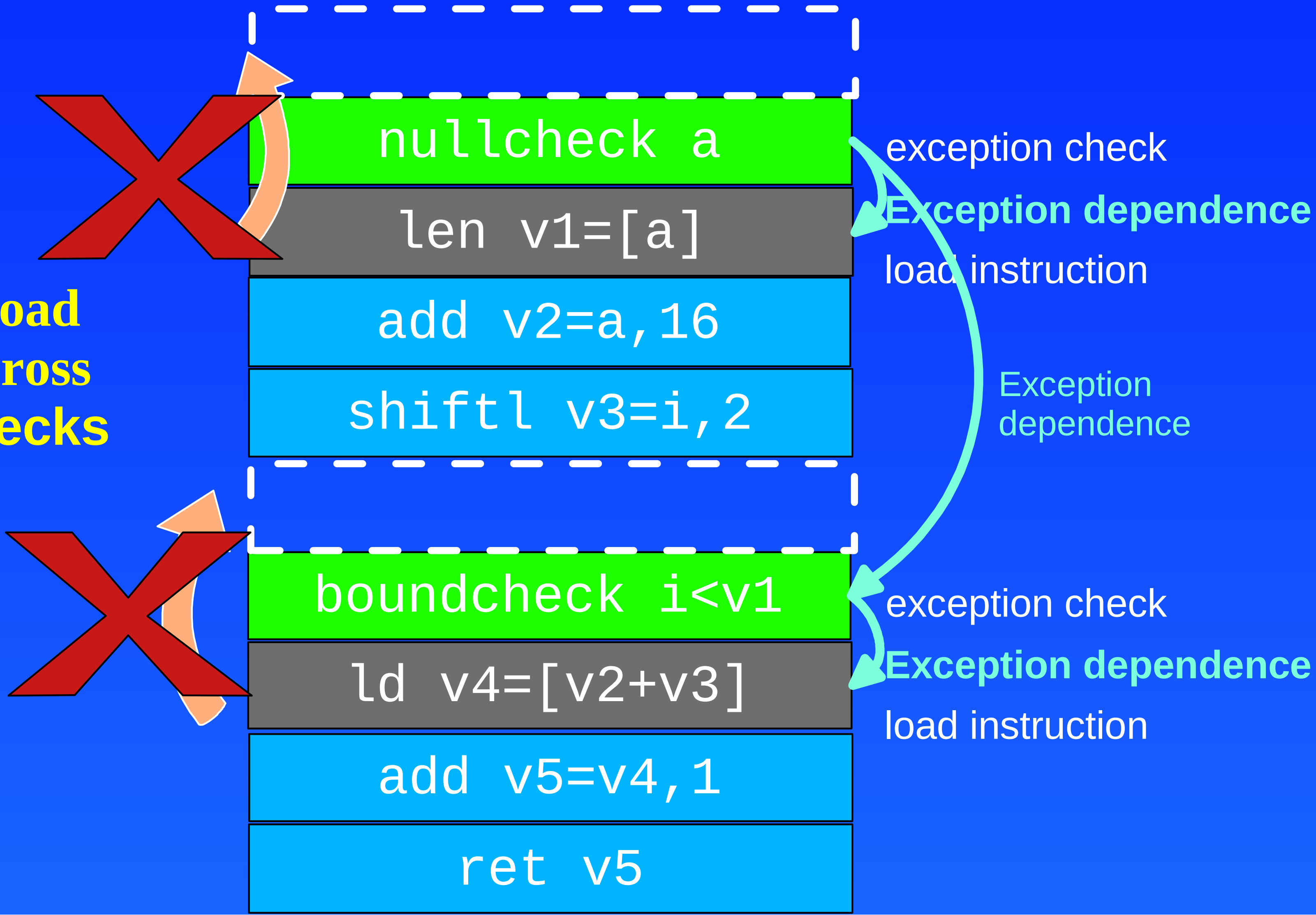
```
iaload
iconst_1
iadd
ireturn
```

**Intermediate Representation (IR)**

| IR | |
|---|---|
| nullcheck a | exception check |
| len v1=[a] | load instruction |
| add v2=a,16 | |
| shiftl v3=i,2 | |
| boundcheck i<v1 | exception check |
| ld v4=[v2+v3] | load instruction |
| add v5=v4,1 | |
| ret v5 | |

# Problems in Java

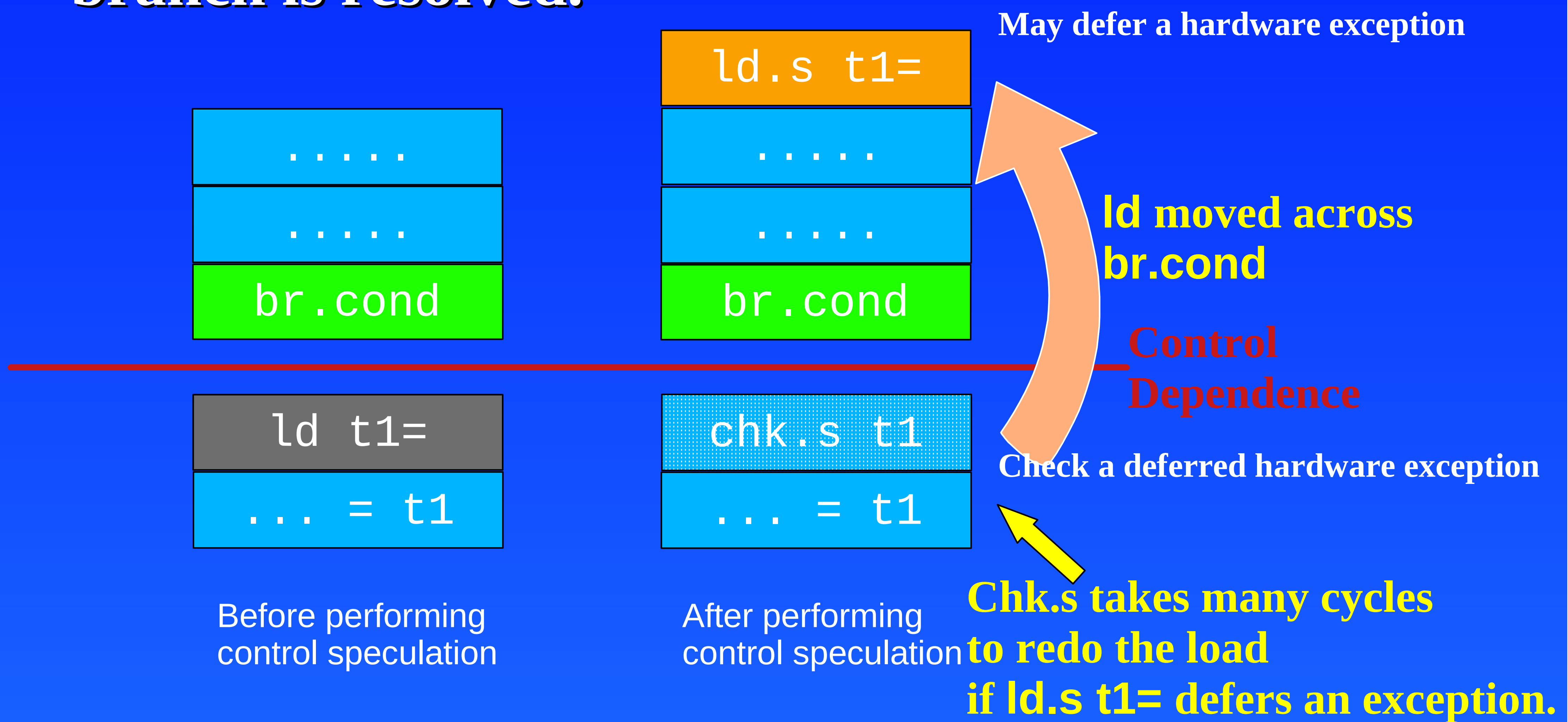
- **An exception dependence between** exception check **and** load **suppresses code motion**




**Cannot move load instructions across exception checks**

# Control Speculation in IA-64

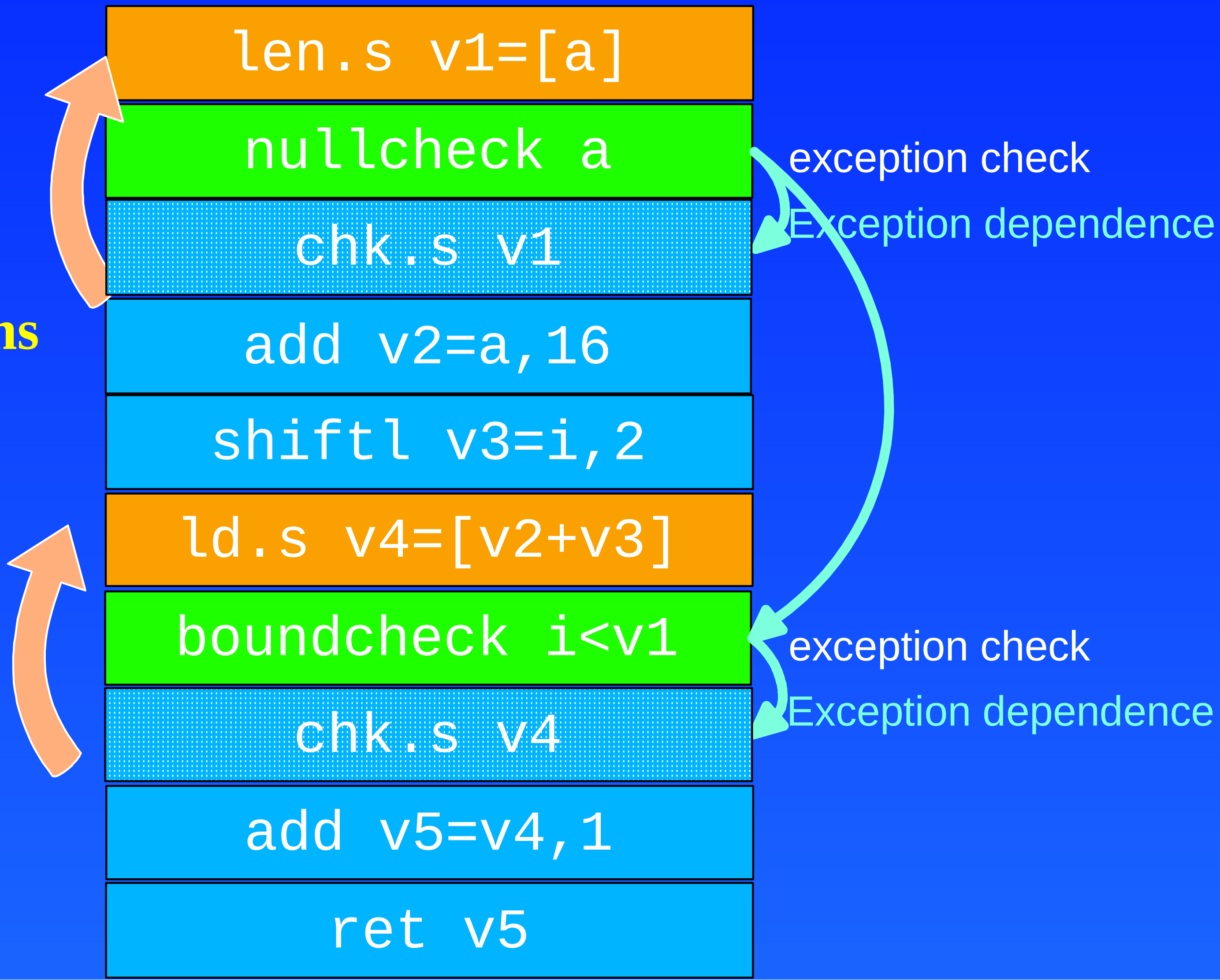
- **An speculative load instruction allows dependant loads to issue a load earlier before the conditional branch is resolved.**



# Our Approach - Exception Speculation

- **Eliminate exception dependence edges from each load**

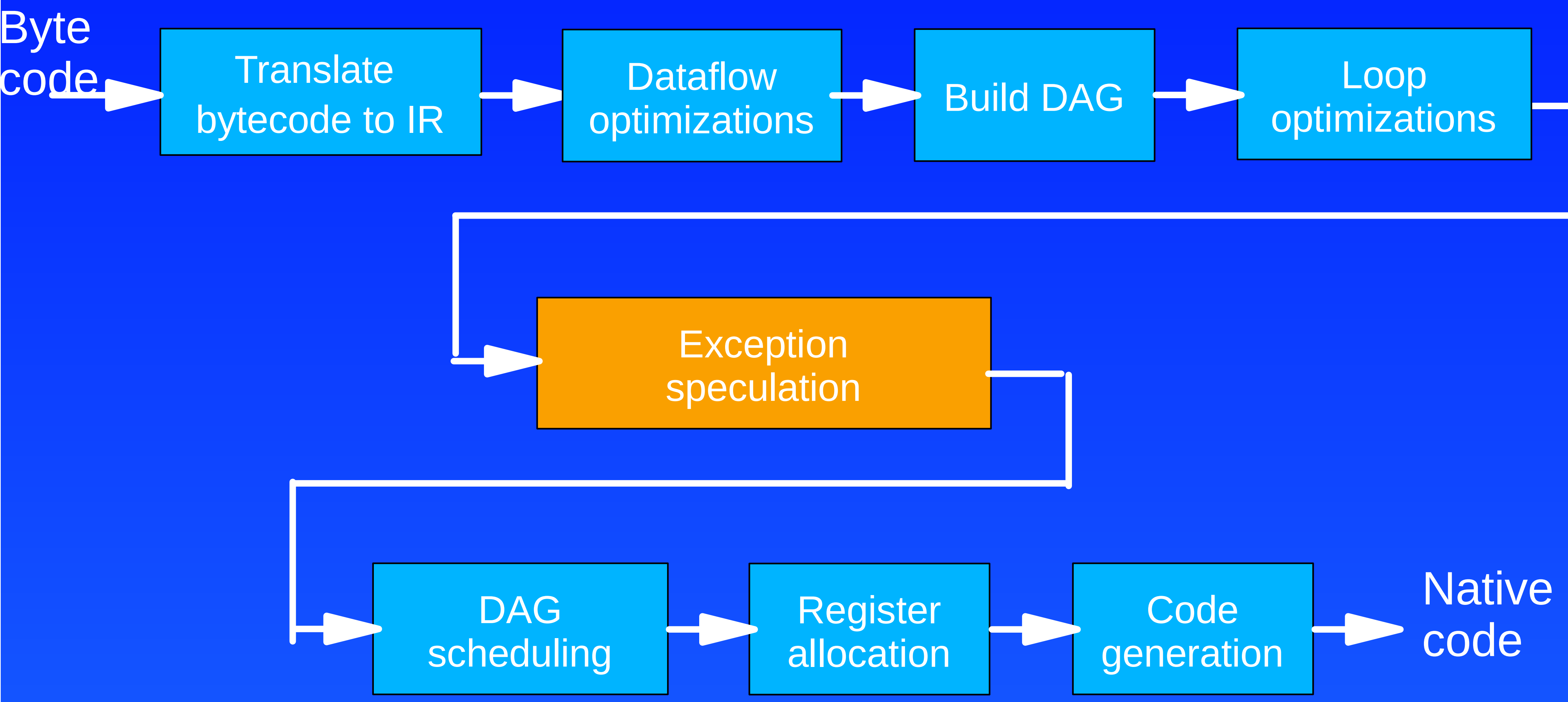
**Move load instructions across exception checks**



# Why We Distinguish Between Control and Exception Speculation

- Reduce the size of IR by not splitting basic blocks
  - We do not handle exception dependence as control dependence.
  - In our experiments, # of basic blocks can be increased by a factor of four without using exception dependence edges.
- Estimate the benefit of exception speculation along the exception dependence edge.
  - The code can be moved speculatively only when it is beneficial on the DAG.

# Where We Perform Exception Speculation



# Algorithm Outline

1. **Decide whether a load can be moved speculatively**
   - ► **When Delay(n) is set only by exception dependence, where *n* is an instruction.**

$$\mathrm{Delay}(n) = \max_{m \in \mathrm{Pred}(n,\mathrm{DAG})} \mathrm{Delay}(m) + \mathrm{Latency}(m)$$

2. **Determine a *speculative chain***
   - ► **Load and the succeeding instructions w/o side effect**
3. **Eliminate and connect exception dependence edges**
   - ► **Restructure a DAG to issue a load earlier.**
4. **Create dependence edges**
   - ► **Maintain edges to preserve the correctness**

# Our DAG for Exception Speculation

- **Before eliminating exception dependence edge**

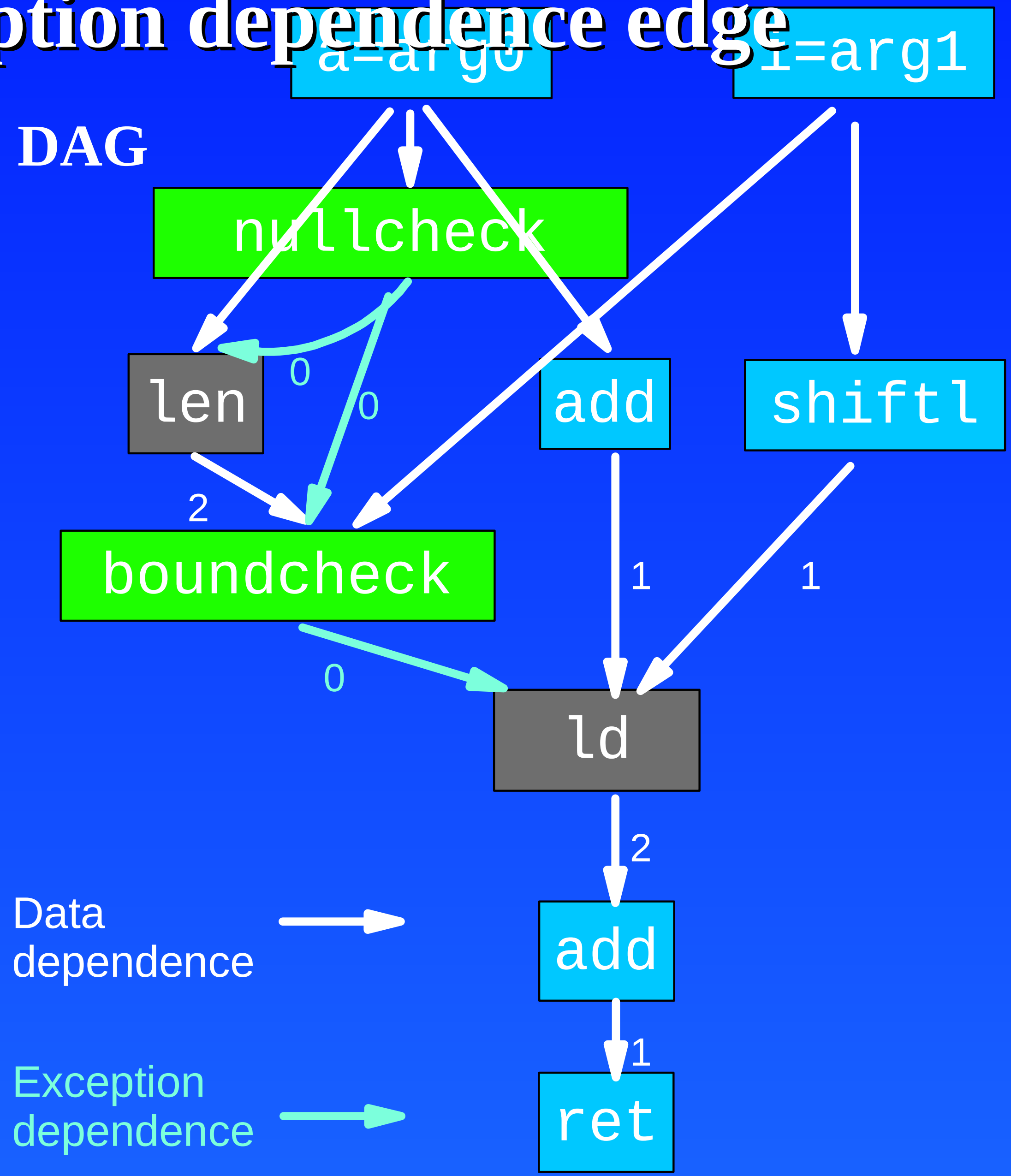
**Java Program**

```
int foo(int a[], int i) {
  return a[i] + 1;
}
```

**Intermediate Representation**

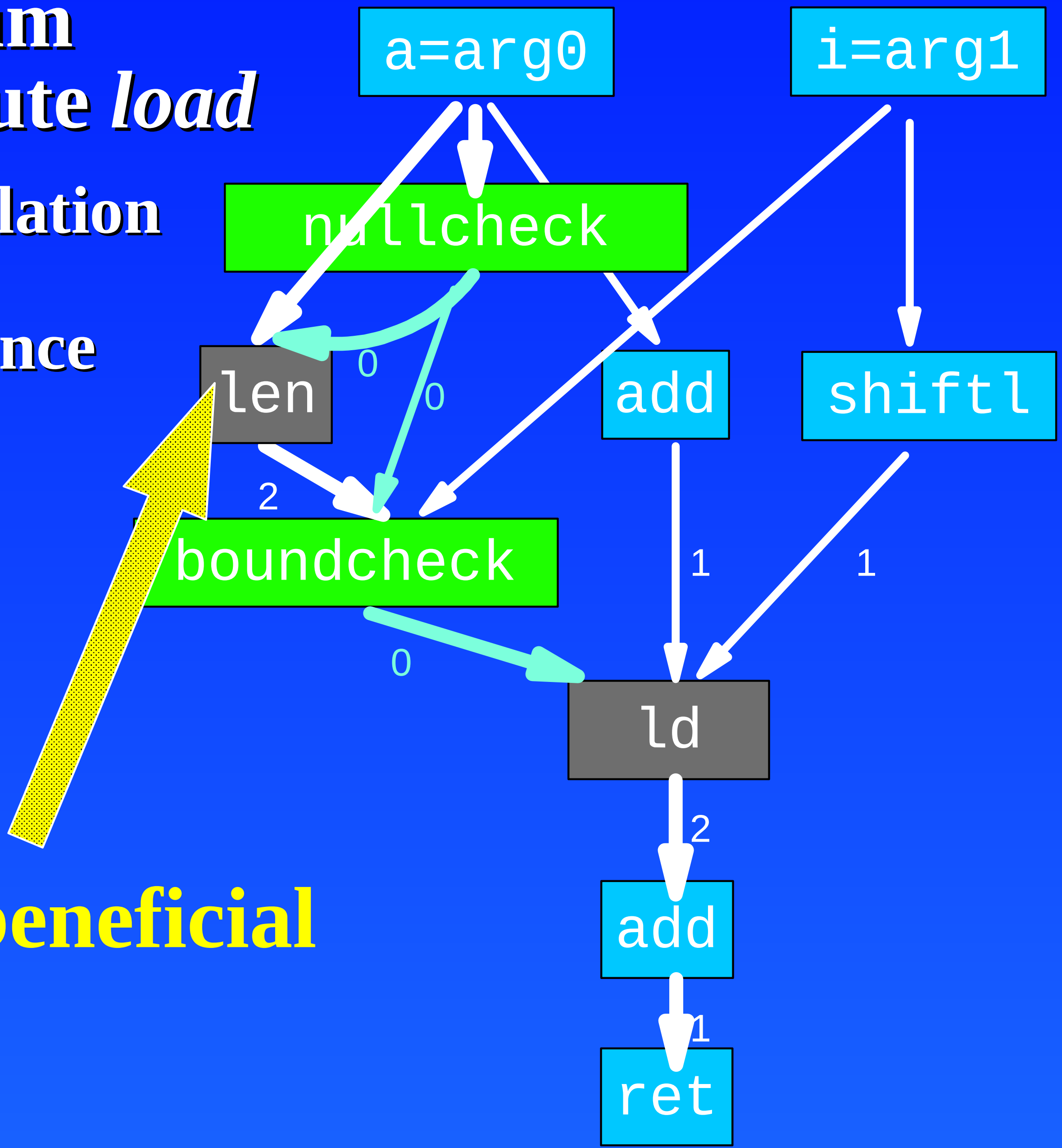
```
nullcheck    a
len          v1 = [a]
add          v2 = a, 16
shiftl       v3 = i, 2
boundcheck   i < v1
ld           v4 = [v2+v3]
add          v5 = v4, 1
ret          v5
```

**DAG**



# Decision to Perform Exception Speculation

- **Calculate the maximum possible delay to execute *load***
  - **Perform exception speculation if the time set by exception dependence is the slowest.**

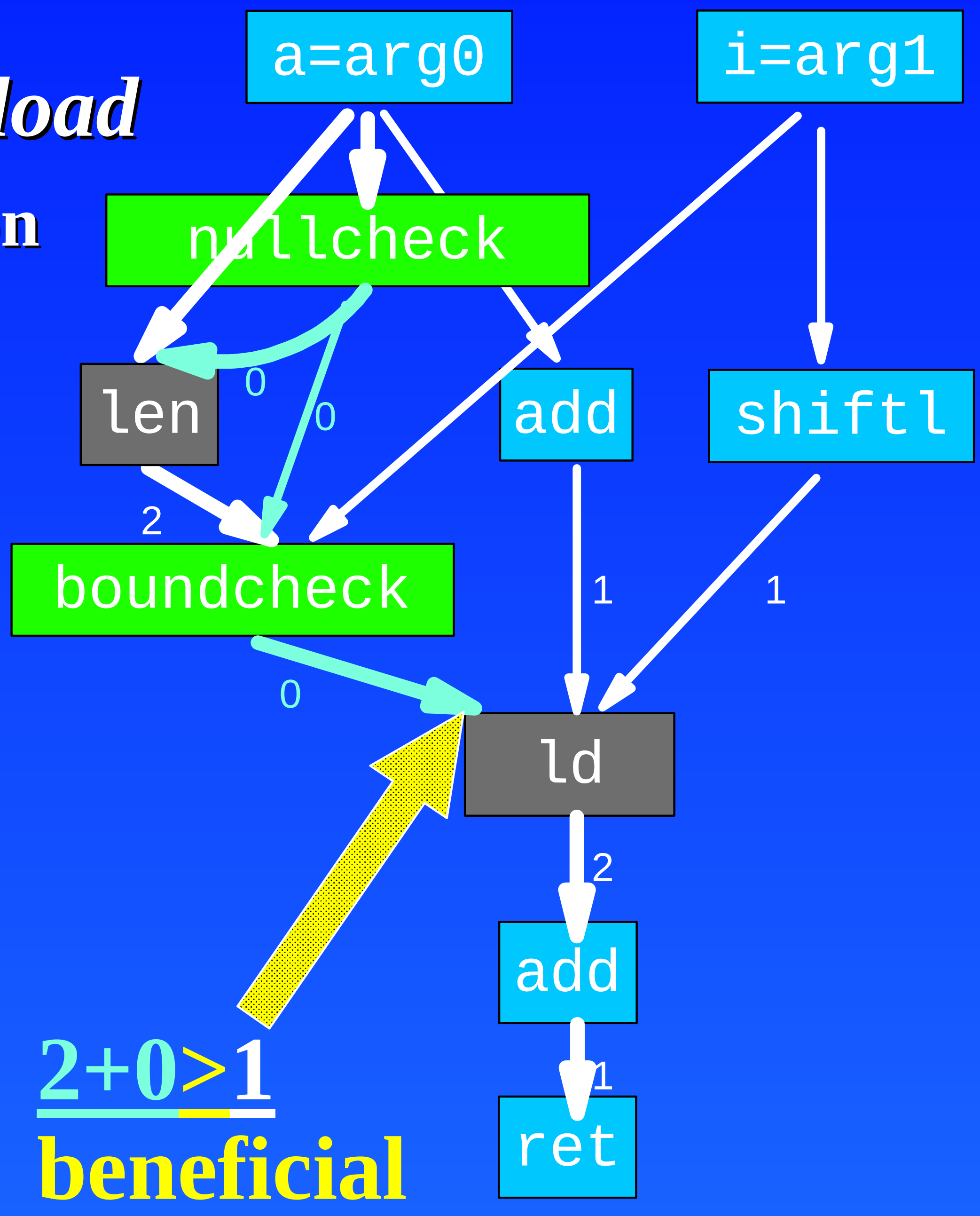



**0 = 0**

**non-beneficial**

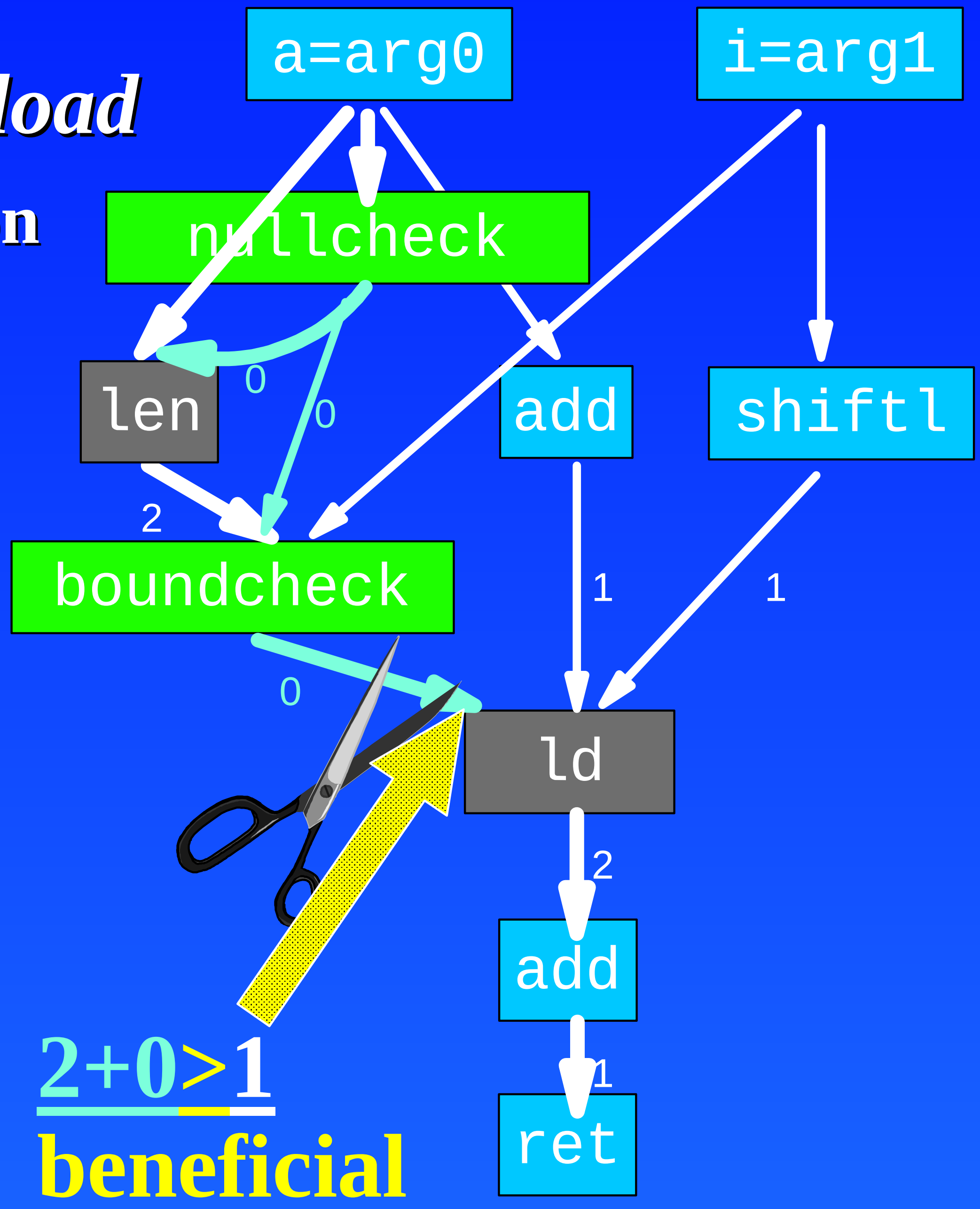
# Decision to Perform Exception Speculation

- **Calculate the maximum possible delay to execute *load***
  - **Perform exception speculation if the time set by exception dependence is the slowest.**



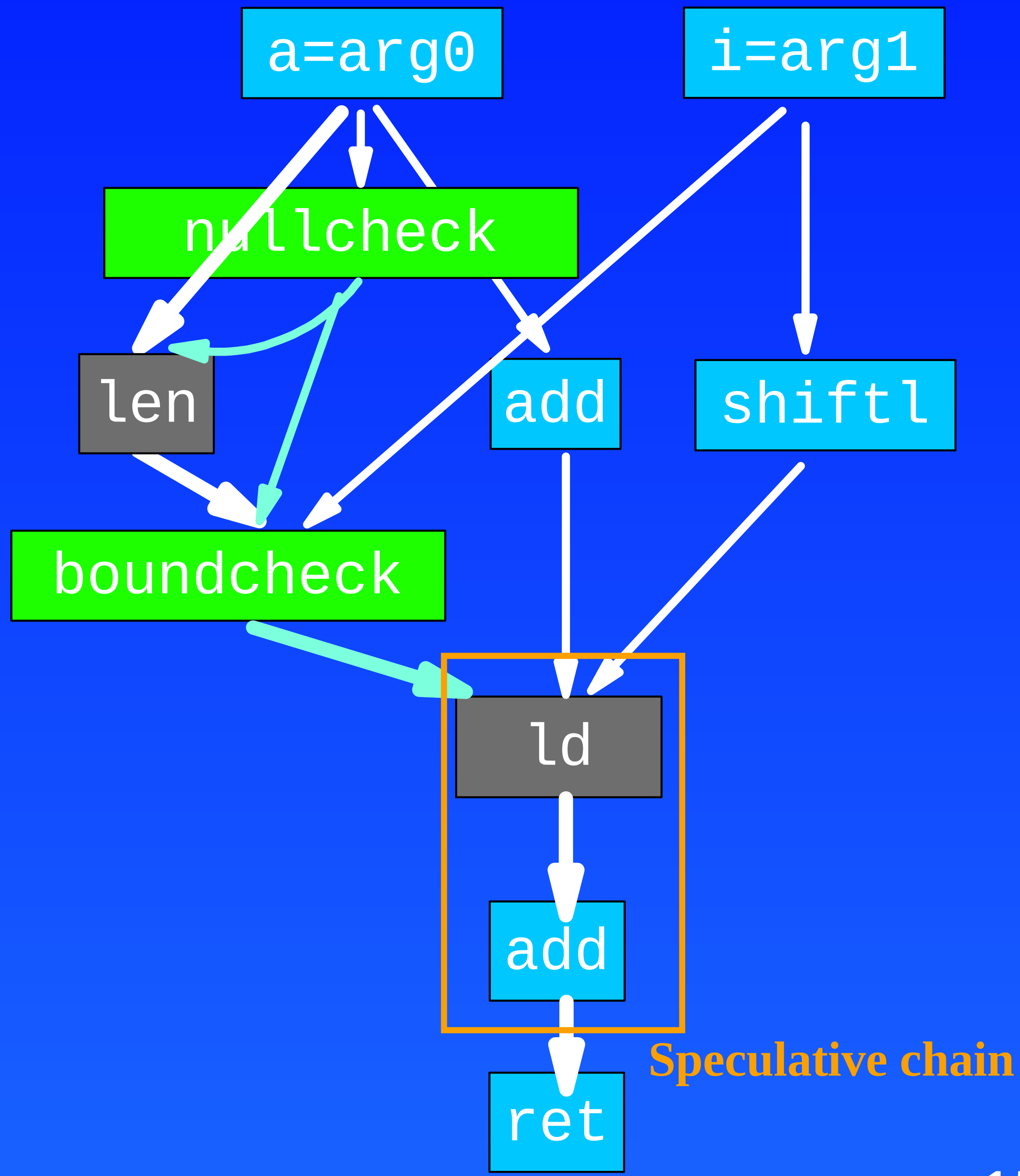
# Decision to Perform Exception Speculation

- **Calculate the maximum possible delay to execute *load***
  - **Perform exception speculation if the time set by exception dependence is the slowest.**



# Determine Speculative Chain

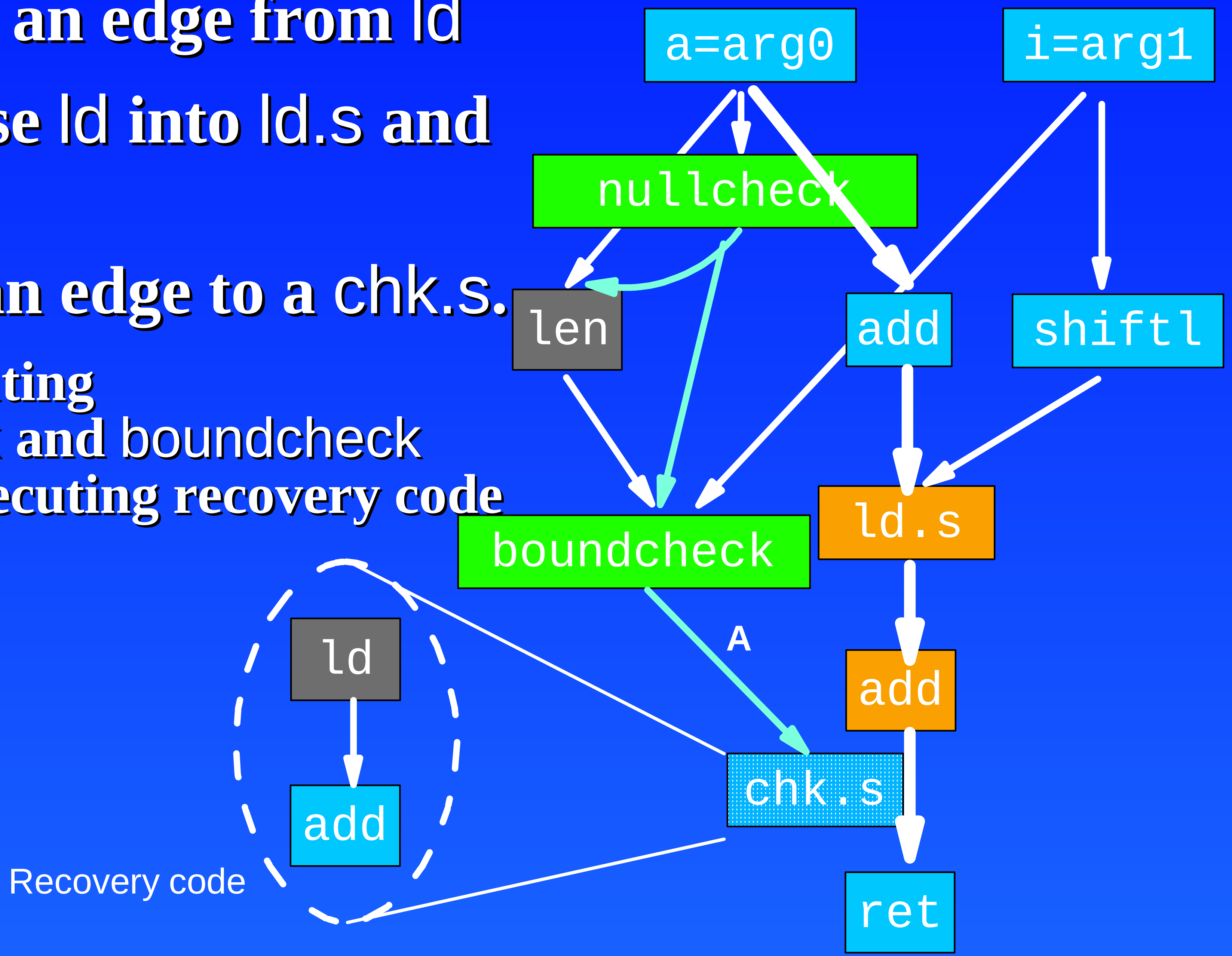
- **Determine a chain of instructions that have no side effects as a *speculative chain***



# Exception Dependence Edge

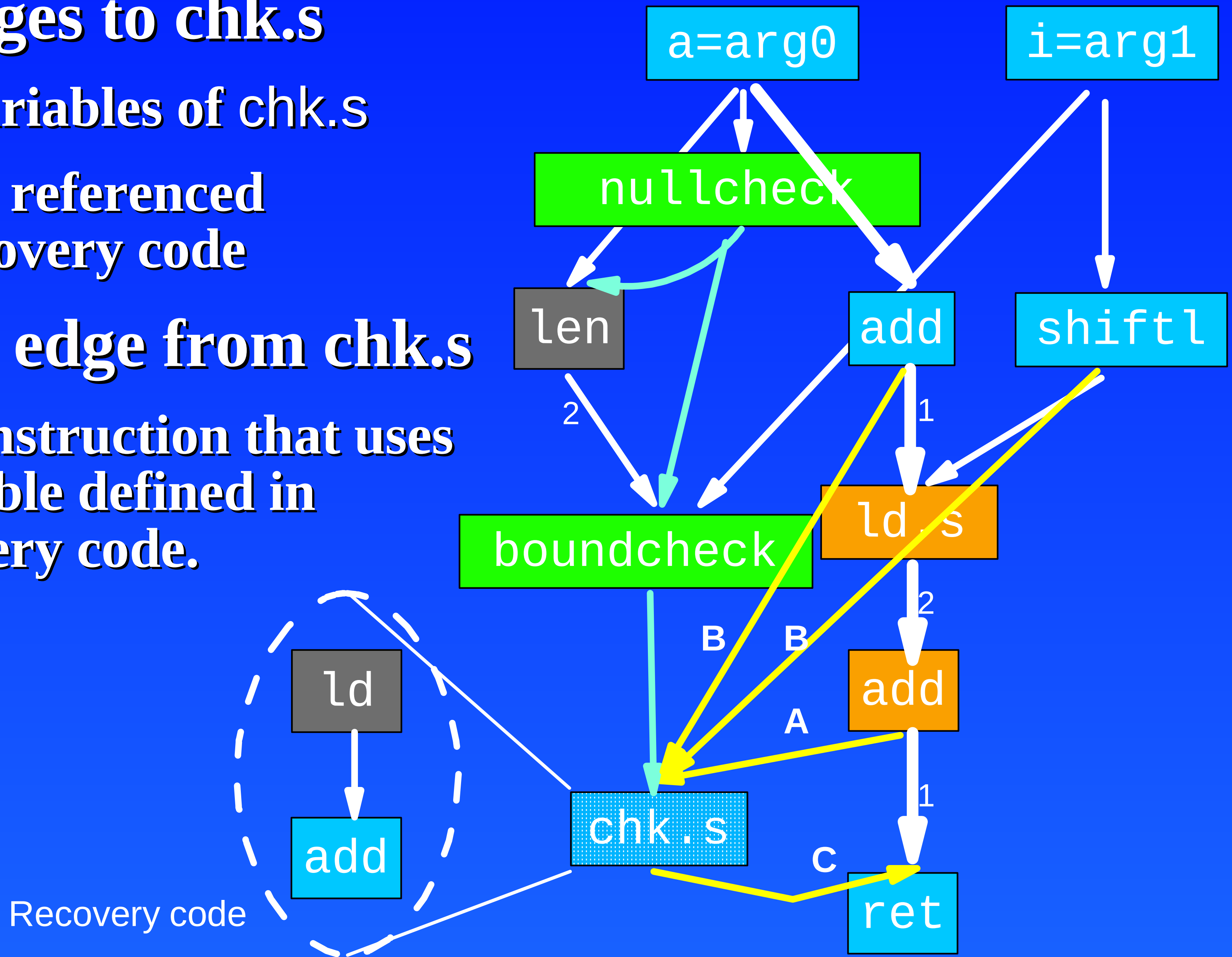
- Eliminate an edge from ld
- Decompose ld into ld.s and chk.s
- Connect an edge to a chk.s.

  A. For executing nullcheck and boundcheck before executing recovery code



# Dependence Edges

- Create edges to chk.s
  - A. Source variables of chk.s
  - B. Variables referenced in the recovery code
- Create an edge from chk.s
  - C. To each instruction that uses any variable defined in the recovery code.

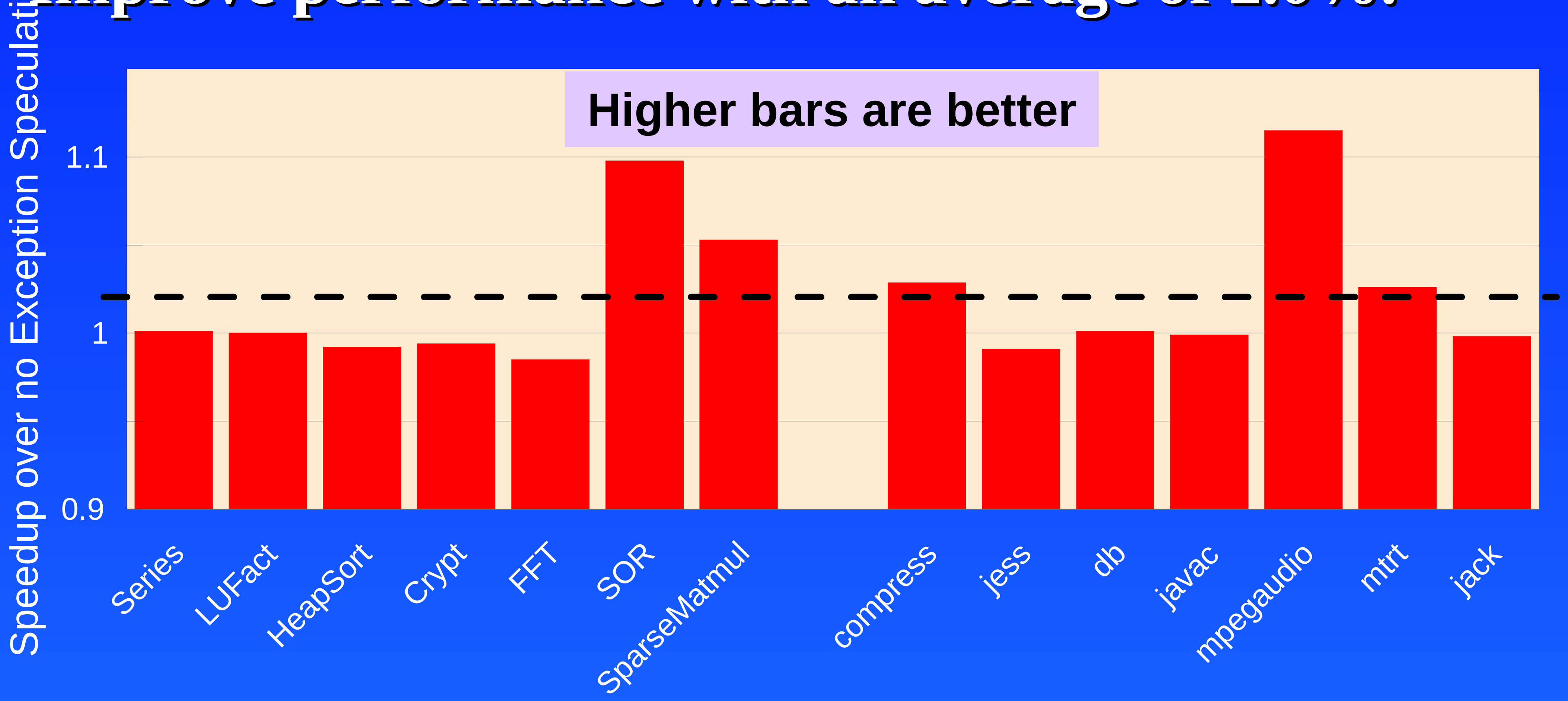


# Experimental Results

- Measurements for:
  - Performance improvement
  - Code size expansion
- Benchmarks
  - Java Grande Benchmark Version 2.0
  - SPECjvm98
- Environments
  - IBM Developers Kit for IA-64, Java Technology Edition, 1.3
  - 2-way 800MHz Itanium with 2GB memory
  - Windows XP Advanced Server

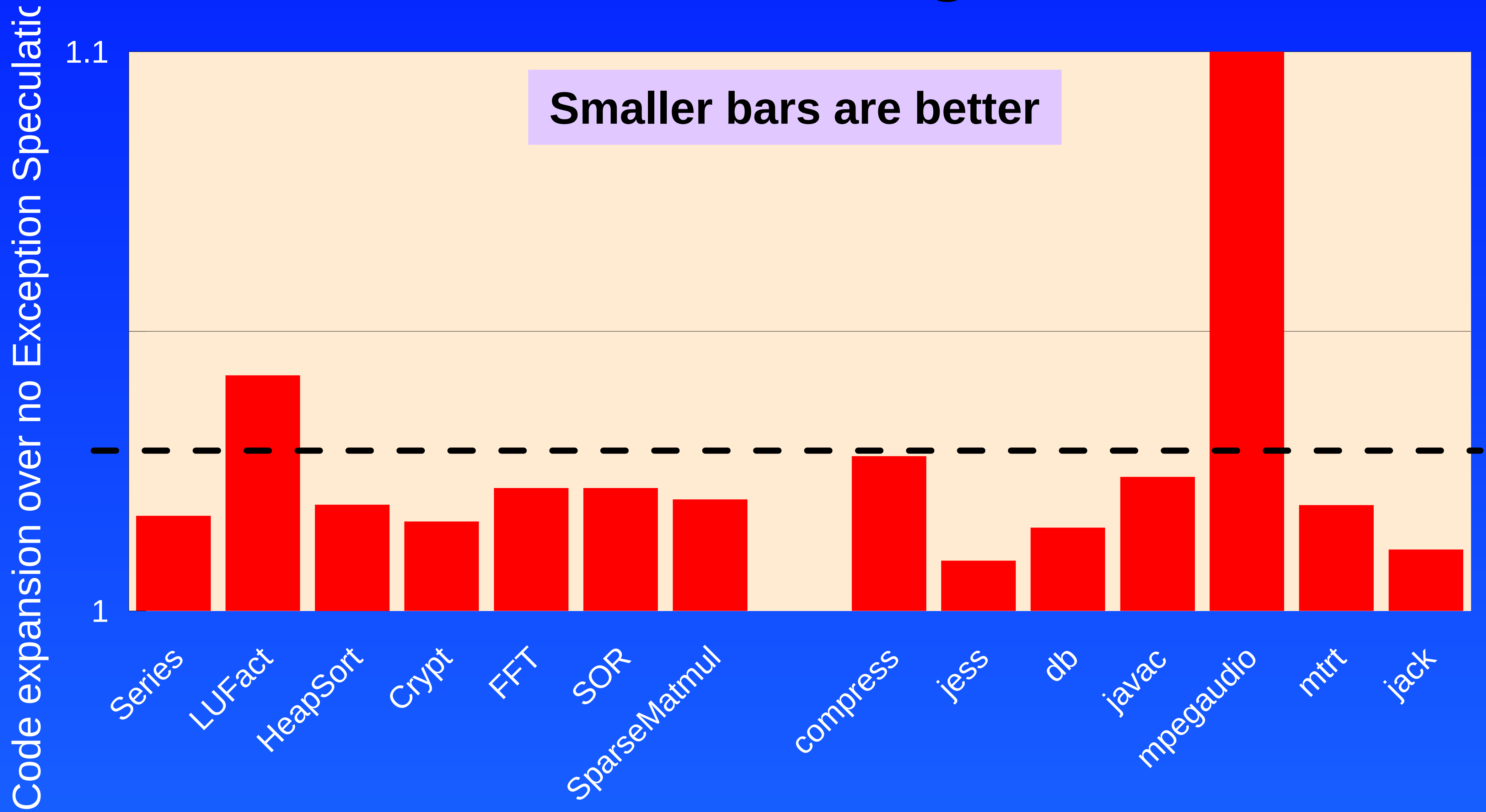
# Performance Improvement

- Exception speculation is effective in programs with many array accesses
- Improve performance with an average of 2.0%.



# Code Size Expansion

- **Increase code size with an average of 2.6%.**



# Summary

- **Propose a new solution "*exception speculation*"**
  - **Eliminate constraint of a load instruction by exception dependences on a DAG representation.**
  - **Perform speculative code motion based on cost-benefit analysis.**
- **Show the effectiveness using a production Java JIT compiler**

# Thanks !!

- **Let's take a coffee break.**

Just In

Time...