# Generative Operational Semantics for Relaxed Memory Models

Radha Jagadeesan Corin Pitcher **James Riely**

School of Computing
DePaul University

ESOP 2010

# An operational semantics for concurrency

- Message passing concurrency well understood
  - operational/denotational models, equivalence/order relations sound type systems, proof systems, etc
- Shared-memory concurrency well understood, assuming
  - sequentially consistent execution, or
  - data race free programs
- Relaxed models used in practice
  - Compiler flexibility (source, JIT, instruction decoder/scheduler)
  - Efficiency, lock free algorithms
- Relaxed models not well understood
  - Goal: novel type system for relaxed model
  - This paper: operational semantics for soundness proof

# An operational semantics for concurrency

- Message passing concurrency well understood
  - operational/denotational models, equivalence/order relations sound type systems, proof systems, etc
- Shared-memory concurrency well understood, assuming
  - sequentially consistent execution, or
  - data race free programs
- Relaxed models used in practice
  - Compiler flexibility (source, JIT, instruction decoder/scheduler)
  - Efficiency, lock free algorithms
- Relaxed models not well understood
  - Goal: novel type system for relaxed model
  - This paper: operational semantics for soundness proof

# An operational semantics for concurrency

- Message passing concurrency well understood
  - operational/denotational models, equivalence/order relations sound type systems, proof systems, etc
- Shared-memory concurrency well understood, assuming
  - sequentially consistent execution, or
  - data race free programs
- Relaxed models used in practice
  - Compiler flexibility (source, JIT, instruction decoder/scheduler)
  - Efficiency, lock free algorithms
- Relaxed models not well understood
  - Goal: novel type system for relaxed model
  - This paper: operational semantics for soundness proof

# An operational semantics for concurrency

- Message passing concurrency well understood
  - operational/denotational models, equivalence/order relations sound type systems, proof systems, etc
- Shared-memory concurrency well understood, assuming
  - sequentially consistent execution, or
  - data race free programs
- Relaxed models used in practice
  - Compiler flexibility (source, JIT, instruction decoder/scheduler)
  - Efficiency, lock free algorithms
- Relaxed models not well understood
  - Goal: novel type system for relaxed model
  - This paper: operational semantics for soundness proof

# An operational semantics for concurrency

- Message passing concurrency well understood
  - operational/denotational models, equivalence/order relations sound type systems, proof systems, etc
- Shared-memory concurrency well understood, assuming
  - sequentially consistent execution, or
  - data race free programs
- Relaxed models used in practice
  - Compiler flexibility (source, JIT, instruction decoder/scheduler)
  - Efficiency, lock free algorithms
- Relaxed models not well understood
  - Goal: novel type system for relaxed model
  - This paper: operational semantics for soundness proof

# Transformations that occur in relaxed models

- Non-conflict reordering (conflict = same location + write)

```
p.f=0        p.g=1
p.g=1   →    p.f=0
```

- Redundant read elimination

```
x=p.f            x=p.f
p.g=1            p.g=1
y=p.f       →    return x
return y
```

- Roach motel

```
x=p.f              k.acquire()
k.acquire()   →    x=p.f
```

# Transformations that occur in relaxed models

- Non-conflict reordering (conflict = same location + write)

```
p.f=0      p.g=1
p.g=1  →   p.f=0
```

- Redundant read elimination

```
x=p.f           x=p.f
p.g=1           p.g=1
y=p.f       →   return x
return y
```

- Roach motel

```
x=p.f            k.acquire()
k.acquire()  →   x=p.f
```

# Transformations that occur in relaxed models

- Non-conflict reordering (conflict $=$ same location $+$ write)

```
p.f=0     p.g=1
p.g=1  →  p.f=0
```

- Redundant read elimination

```
x=p.f          x=p.f
p.g=1          p.g=1
y=p.f       →  return x
return y
```

- Roach motel

```
x=p.f             k.acquire()
k.acquire()   →   x=p.f
```

# Concurrency and program transformation

- Transformation correct: **no new behavior**
- Expect `p.h` incremented at most once (Dijkstra, 1965)

*write g, read f*            *write f, read g*

```
p.g = 1             p.f = 1
x = p.f             y = p.g
if(x == 0)          if(y == 0)
  p.h ++              p.h ++
```

- Non-conflict reordering

```
x = p.f             p.f = 1
p.g = 1             y = p.g
if(x == 0)          if(y == 0)
  p.h ++              p.h ++
```

# Concurrency and program transformation

- Transformation correct: **no new behavior**
- Expect `p.h` incremented at most once (Dijkstra, 1965)

```
write f, read g      write f, read g
x = p.f              p.f = 1
p.g = 1              y = p.g
if(x == 0)           if(y == 0)
  p.h ++               p.h ++
```

- Non-conflict reordering

```
x = p.f              p.f = 1
p.g = 1              y = p.g
if(x == 0)           if(y == 0)
  p.h ++               p.h ++
```

# Concurrency and program transformation

- Transformation correct: **no new behavior**
- Expect `p.h` incremented at most once (Dijkstra, 1965)

```
write g, read f      write f, read g
p.g = 1              p.f = 1
x = p.f              y = p.g
if(x == 0)           if(y == 0)
  p.h ++               p.h ++
```

- Non-conflict reordering

```
x = p.f              p.f = 1
p.g = 1              y = p.g
if(x == 0)           if(y == 0)
  p.h ++               p.h ++
```

# Concurrency and program transformation

- Transformation correct: **no new behavior**
- Expect `p.h` incremented at most once (Dijkstra, 1965)

```
write g, read f       write f, read g
p.g = 1               p.f = 1
x = p.f               y = p.g
if(x == 0)            if(y == 0)
  p.h ++                p.h ++
```

- Non-conflict reordering

```
 p.f=0     p.g=0     p.h=0

x = p.f               p.f = 1
p.g = 1               y = p.g
if(x == 0)            if(y == 0)
  p.h ++                p.h ++
```

# Concurrency and program transformation

- Transformation correct: **no new behavior**
- Expect **p.h** incremented at most once (Dijkstra, 1965)

```
write g, read f         write f, read g
p.g = 1                 p.f = 1
x = p.f                 y = p.g
if(x == 0)              if(y == 0)
  p.h ++                  p.h ++
```

- Non-conflict reordering

```
 p.f=0      p.g=0      p.h=0

0 = p.f                 p.f = 1
p.g = 1                 y = p.g
if(0 == 0)              if(y == 0)
  p.h ++                  p.h ++
```

# Concurrency and program transformation

- Transformation correct: **no new behavior**
- Expect `p.h` incremented at most once (Dijkstra, 1965)

```
write g, read f         write f, read g
p.g = 1                 p.f = 1
x = p.f                 y = p.g
if(x == 0)              if(y == 0)
  p.h ++                  p.h ++
```

- Non-conflict reordering

```
  p.f=1      p.g=0      p.h=0

0 = p.f                 p.f = 1
p.g = 1                 y = p.g
if(0 == 0)              if(y == 0)
  p.h ++                  p.h ++
```

# Concurrency and program transformation

- Transformation correct: **no new behavior**
- Expect `p.h` incremented at most once (Dijkstra, 1965)

```
write g, read f      write f, read g
p.g = 1              p.f = 1
x = p.f              y = p.g
if(x == 0)           if(y == 0)
  p.h ++               p.h ++
```

- Non-conflict reordering

| p.f=1 | p.g=0 | p.h=0 |
|---|---|---|

```
0 = p.f              p.f = 1
p.g = 1              0 = p.g
if(0 == 0)           if(0 == 0)
  p.h ++               p.h ++
```

# Concurrency and program transformation

- Transformation correct: **no new behavior**
- Expect `p.h` incremented at most once (Dijkstra, 1965)

```
write g, read f        write f, read g
p.g = 1                p.f = 1
x = p.f                y = p.g
if(x == 0)             if(y == 0)
  p.h ++                 p.h ++
```

- Non-conflict reordering

```
p.f=1      p.g=0      p.h=1

0 = p.f               p.f = 1
p.g = 1               0 = p.g
if(0 == 0)            if(0 == 0)
  p.h ++                p.h ++
```

# Concurrency and program transformation

- Transformation correct: **no new behavior**
- Expect **`p.h`** incremented at most once (Dijkstra, 1965)

*write g, read f*      *write f, read g*

```
p.g = 1          p.f = 1
x = p.f          y = p.g
if(x == 0)       if(y == 0)
  p.h ++           p.h ++
```

- Non-conflict reordering

**p.f**=1    **p.g**=1    **p.h**=1

```
0 = p.f          p.f = 1
p.g = 1          0 = p.g
if(0 == 0)       if(0 == 0)
  p.h ++           p.h ++
```

# Concurrency and program transformation

- Transformation correct: **no new behavior**
- Expect `p.h` incremented at most once (Dijkstra, 1965)

```
write g, read f     write f, read g
p.g = 1             p.f = 1
x = p.f             y = p.g
if(x == 0)          if(y == 0)
  p.h ++              p.h ++
```

- Non-conflict reordering

| p.f=1 | p.g=1 | p.h=2 |
|---|---|---|

```
0 = p.f             p.f = 1
p.g = 1             0 = p.g
if(0 == 0)          if(0 == 0)
  p.h ++              p.h ++
```

1. Background
    - Sequential Consistency
    - Data Race Free Model
    - Java Memory Model

2. Speculative semantics
    - Empirical and speculative actions
    - Desirable executions allowed
    - Undesirable executions prevented

3. Summary of results
    - Relation to Java Memory Model
    - Simulation precongruence

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

Program: *write f twice* *read f twice*

```
p.f = 0      x = p.f
p.f = 1      y = p.f
return       return(x,y)
```

Memory:

Threads:

```
p.f = 0      x = p.f
p.f = 1      y = p.f
return       return(x,y)
```

`return (1,1)` possible
`return (0,1)` possible
`return (0,0)` possible
`return (1,0)` impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

Program: *write f twice* *read f twice*

```
p.f = 0        x = p.f
p.f = 1        y = p.f
return         return(x,y)
```

Memory:

Threads:

```
p.f = 0        x = p.f
p.f = 1        y = p.f
return         return(x,y)
```

`return (1,1)` possible
`return (0,1)` possible
`return (0,0)` possible
`return (1,0)` impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

```
Program:    write f twice      read f twice
            p.f = 0            x = p.f
            p.f = 1            y = p.f
            return             return(x,y)
```

```
Memory:     p.f=0

Threads:    p.f = 0            x = p.f
            p.f = 1            y = p.f
            return             return(x,y)
```

`return (1,1)` possible
`return (0,1)` possible
`return (0,0)` possible
`return (1,0)` impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

Program: *write f twice* *read f twice*

```
p.f = 0         x = p.f
p.f = 1         y = p.f
return          return(x,y)
```

Memory: `p.f=0`

Threads:

```
p.f = 0         x = p.f
p.f = 1         y = p.f
return          return(x,y)
```

`return (1,1)` possible
`return (0,1)` possible
`return (0,0)` possible
`return (1,0)` impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

Program: *write f twice* *read f twice*

```
p.f = 0       x = p.f
p.f = 1       y = p.f
return        return(x,y)
```

Memory: **p.f**=**1**

Threads:

```
p.f = 0       x = p.f
p.f = 1       y = p.f
return        return(x,y)
```

`return (1,1)` possible
`return (0,1)` possible
`return (0,0)` possible
`return (1,0)` impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order  
Ops of individual threads appear in program order  
(Lamport 1977)

```
Program:   write f twice     read f twice
           p.f = 0           x = p.f
           p.f = 1           y = p.f
           return            return(x,y)
```

```
Memory:    p.f=1

Threads:   p.f = 0           1 = p.f
           p.f = 1           y = p.f
           return            return(1,y)
```

`return (1,1)` possible  
`return (0,1)` possible  
`return (0,0)` possible  
`return (1,0)` impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

Program: *write f twice* *read f twice*

```
p.f = 0        x = p.f
p.f = 1        y = p.f
return         return(x,y)
```

Memory: p.f=1

Threads:

```
p.f = 0        1 = p.f
p.f = 1        1 = p.f
return         return(1,1)
```

`return (1,1)` possible
`return (0,1)` possible
`return (0,0)` possible
`return (1,0)` impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

Program:

```
write f twice      read f twice
p.f = 0            x = p.f
p.f = 1            y = p.f
return             return(x,y)
```

Memory: p.f=0

Threads:

```
p.f = 0            x = p.f
p.f = 1            y = p.f
return             return(x,y)
```

`return (1,1)` possible
`return (0,1)` possible
`return (0,0)` possible
`return (1,0)` impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

```
Program:   write f twice    read f twice
           p.f = 0          x = p.f
           p.f = 1          y = p.f
           return           return(x,y)
```

```
Memory:    p.f=0

Threads:   p.f = 0          x = p.f
           p.f = 1          y = p.f
           return           return(x,y)
```

`return (1,1)` possible
`return (0,1)` possible
`return (0,0)` possible
`return (1,0)` impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

```
Program:   write f twice      read f twice
           p.f = 0            x = p.f
           p.f = 1            y = p.f
           return             return(x,y)
```

```
Memory:    p.f=0

Threads:   p.f = 0            x = p.f
           p.f = 1            y = p.f
           return             return(x,y)
```

**return (1,1)** possible
return (0,1) possible
return (0,0) possible
return (1,0) impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

```
Program:   write f twice     read f twice
           p.f = 0           x = p.f
           p.f = 1           y = p.f
           return            return(x,y)
```

```
Memory:    p.f=1

Threads:   p.f = 0           0 = p.f
           p.f = 1           y = p.f
           return            return(0,y)
```

`return (1,1)` possible
`return (0,1)` possible
`return (0,0)` possible
`return (1,0)` impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

Program: *write f twice* *read f twice*

```
p.f = 0         x = p.f
p.f = 1         y = p.f
return          return(x,y)
```

Memory: p.f=1

Threads:

```
p.f = 0         0 = p.f
p.f = 1         y = p.f
return          return(0,y)
```

`return (1,1)` possible
`return (0,1)` possible
`return (0,0)` possible
`return (1,0)` impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

Program: *write f twice* *read f twice*

```
p.f = 0      x = p.f
p.f = 1      y = p.f
return       return(x,y)
```

Memory: `p.f=1`

Threads:

```
p.f = 0      0 = p.f
p.f = 1      1 = p.f
return       return(0,1)
```

`return (1,1)` possible
`return (0,1)` possible
`return (0,0)` possible
`return (1,0)` impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

```
Program:   write f twice      read f twice
           p.f = 0            x = p.f
           p.f = 1            y = p.f
           return             return(x,y)

Memory:    p.f=1

Threads:   p.f = 0            0 = p.f
           p.f = 1            1 = p.f
           return             return(0,1)
```

`return (1,1)` possible
`return (0,1)` possible
`return (0,0)` possible
`return (1,0)` impossible

# Sequential Consistency (SC)

Ops appear to execute in some sequential order
Ops of individual threads appear in program order
(Lamport 1977)

Program: *write f twice* *read f twice*

```
p.f = 0          x = p.f
p.f = 1          y = p.f
return           return(x,y)
```

Memory: `p.f=1`

Threads:

```
p.f = 0          0 = p.f
p.f = 1          1 = p.f
return           return(0,1)
```

`return (1,1)` possible
`return (0,1)` possible
`return (0,0)` possible
`return (1,0)` impossible

# Caching model

Indirection between action and memory

Program:

```
write f twice     read f twice
p.f = 0           x = p.f
p.f = 1           y = p.f
return            return(x,y)
```

Memory: `p.f=0`

Pending actions:

Threads:

```
p.f = 0           x = p.f
p.f = 1           y = p.f
return            return(x,y)
```

`return (1,0)` possible

The execution has a **data race**: conflicting ops not totally ordered

# Caching model

Indirection between action and memory

Program: *write f twice* *read f twice*

```
p.f = 0        x = p.f
p.f = 1        y = p.f
return         return(x,y)
```

Memory: **p.f**=0

Pending actions: **p.f**=0

Threads:

```
p.f = 0        x = p.f
p.f = 1        y = p.f
return         return(x,y)
```

`return (1,0)` possible

The execution has a **data race**: conflicting ops not totally ordered

# Caching model

Indirection between action and memory

Program: *write f twice* *read f twice*

```
p.f = 0        x = p.f
p.f = 1        y = p.f
return         return(x,y)
```

Memory: `p.f=0`

Pending actions: `p.f=0` `p.f=1`

Threads:

```
p.f = 0        x = p.f
p.f = 1        y = p.f
return         return(x,y)
```

`return (1,0)` possible

The execution has a **data race**: conflicting ops not totally ordered

# Caching model

Indirection between action and memory

Program:

| *write f twice* | *read f twice* |
|---|---|
| `p.f = 0` | `x = p.f` |
| `p.f = 1` | `y = p.f` |
| `return` | `return(x,y)` |

Memory: **p.f**=**1**

Pending actions: **p.f**=**0**

Threads:

| | |
|---|---|
| `p.f = 0` | `x = p.f` |
| `p.f = 1` | `y = p.f` |
| `return` | `return(x,y)` |

`return (1,0)` possible

The execution has a **data race**: conflicting ops not totally ordered

# Caching model

Indirection between action and memory

Program:

| *write f twice* | *read f twice* |
|---|---|
| `p.f = 0` | `x = p.f` |
| `p.f = 1` | `y = p.f` |
| `return` | `return(x,y)` |

Memory: **`p.f`**=**`1`**

Pending actions: <u>**`p.f`**=**`0`**</u>

Threads:

| | |
|---|---|
| `p.f = 0` | `1 = p.f` |
| `p.f = 1` | `y = p.f` |
| `return` | `return(1,y)` |

`return (1,0)` possible

The execution has a **data race**: conflicting ops not totally ordered

# Caching model

Indirection between action and memory

Program:

```
write f twice      read f twice
p.f = 0            x = p.f
p.f = 1            y = p.f
return             return(x,y)
```

Memory: p.f=0

Pending actions:

Threads:

```
p.f = 0            1 = p.f
p.f = 1            y = p.f
return             return(1,y)
```

`return (1,0)` possible

The execution has a **data race**: conflicting ops not totally ordered

# Caching model

Indirection between action and memory

Program: *write f twice* *read f twice*

```
p.f = 0        x = p.f
p.f = 1        y = p.f
return         return(x,y)
```

Memory: **p.f**=0

Pending actions:

Threads:

```
p.f = 0        1 = p.f
p.f = 1        0 = p.f
return         return(1,0)
```

**`return (1,0)`** possible

The execution has a **data race**: conflicting ops not totally ordered

# Data-Race Free (DRF) Semantics

- DRF programs: SC execution
- Programs with races: no comment
- Ok for C++ (Boehm and Adve, 2008)
  - no benign races
  - no safety guarantees

# Java

- Defines semantics for programs with races (type safety)
- **Defined** (Gosling, Joy, Steele, 1996)
  Caching semantics with “prescient reads”
- **Criticized** (Pugh 1999)
  - invalidates redundant read elimination

```
x=p.f        x=p.f
y=q.f   ↛    y=q.f
z=p.f        z=x
```

  - invalidates non-conflict reordering!

```
x=p.g        y=p.f
y=p.f   ↛    x=p.g
z=q.f        z=q.f
p.x=1        p.x=1
```

- **Replaced** by JMM (JSR 133, 2004)
  Semantics based on series of executions,
  each “committing” a data race

# Java

- Defines semantics for programs with races (type safety)
- **Defined** (Gosling, Joy, Steele, 1996)
  Caching semantics with “prescient reads”
- **Criticized** (Pugh 1999)
  - invalidates redundant read elimination

```
x=p.f         x=p.f
y=q.f   ↛     y=q.f
z=p.f         z=x
```

  - invalidates non-conflict reordering!

```
x=p.g         y=p.f
y=p.f   ↛     x=p.g
z=q.f         z=q.f
p.x=1         p.x=1
```

- **Replaced** by JMM (JSR 133, 2004)
  Semantics based on series of executions,
  each “committing” a data race

# Java

- Defines semantics for programs with races (type safety)
- **Defined** (Gosling, Joy, Steele, 1996)
  Caching semantics with “prescient reads”
- **Criticized** (Pugh 1999)
  - invalidates redundant read elimination

```
x=p.f          x=p.f
y=q.f    ↛     y=q.f
z=p.f          z=x
```

  - invalidates non-conflict reordering!

```
x=p.g          y=p.f
y=p.f    ↛     x=p.g
z=q.f          z=q.f
p.x=1          p.x=1
```

- **Replaced** by JMM (JSR 133, 2004)
  Semantics based on series of executions,
  each “committing” a data race

# Java

- Defines semantics for programs with races (type safety)
- **Defined** (Gosling, Joy, Steele, 1996)
  Caching semantics with “prescient reads”
- **Criticized** (Pugh 1999)
  - invalidates redundant read elimination

```
x=p.f       x=p.f
y=q.f   ↛   y=q.f
z=p.f       z=x
```

  - invalidates non-conflict reordering!

```
x=p.g       y=p.f
y=p.f   ↛   x=p.g
z=q.f       z=q.f
p.x=1       p.x=1
```

- **Replaced** by JMM (JSR 133, 2004)
  Semantics based on series of executions,
  each “committing” a data race

# Prescient read: seeing the future

- Caching not enough
- Program A:

```
copy f to g     read g, write f
x = p.f         y = p.g
p.g = x         p.f = 1
return x        return y
```

- In SC semantics, `return 1` impossible
- Can result from non-conflict reordering

```
Memory:    p.f=0        p.g=0

Threads:   x = p.f      p.f = 1
           p.g = x      y = p.g
           return x     return y
```

# Prescient read: seeing the future

- Caching not enough
- Program A:

```
copy f to g      read g, write f
x = p.f          y = p.g
p.g = x          p.f = 1
return x         return y
```

- In SC semantics, `return 1` impossible
- Can result from non-conflict reordering

```
Memory:    p.f=0        p.g=0

Threads:   x = p.f      p.f = 1
           p.g = x      y = p.g
           return x     return y
```

# Prescient read: seeing the future

- Caching not enough
- Program A:

```
copy f to g        write f, read g
x = p.f            p.f = 1
p.g = x            y = p.g
return x           return y
```

- In SC semantics, `return 1` impossible
- Can result from non-conflict reordering

```
Memory:   p.f=0          p.g=0

Threads:  x = p.f        p.f = 1
          p.g = x        y = p.g
          return x       return y
```

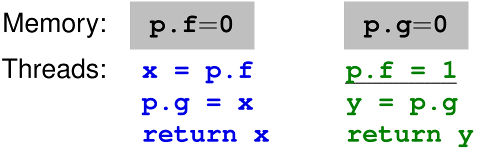
# Prescient read: seeing the future

- Caching not enough
- Program A:

*copy f to g* | *read g, write f*

```
x = p.f          y = p.g
p.g = x          p.f = 1
return x         return y
```

- In SC semantics, `return 1` impossible
- Can result from non-conflict reordering



Memory:

```
p.f=0            p.g=0
```

Threads:

```
x = p.f          p.f = 1
p.g = x          y = p.g
return x         return y
```

# Prescient read: seeing the future

- Caching not enough
- Program A:

| *copy f to g* | *read g, write f* |
|---|---|
| `x = p.f` | `y = p.g` |
| `p.g = x` | `p.f = 1` |
| `return x` | `return y` |

- In SC semantics, `return 1` impossible
- Can result from non-conflict reordering

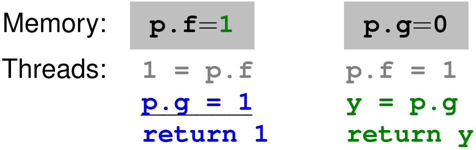
|  |  |  |
|---|---|---|
| Memory: | `p.f`=`1` | `p.g`=`0` |
| Threads: | `x = p.f`<br>`p.g = x`<br>`return x` | `p.f = 1`<br>`y = p.g`<br>`return y` |

# Prescient read: seeing the future

- Caching not enough
- Program A:

```
copy f to g     read g, write f
x = p.f         y = p.g
p.g = x         p.f = 1
return x        return y
```

- In SC semantics, `return 1` impossible
- Can result from non-conflict reordering



```
Memory:    p.f=1         p.g=0
Threads:   1 = p.f       p.f = 1
           p.g = 1       y = p.g
           return 1      return y
```

# Prescient read: seeing the future

- Caching not enough
- Program A:

| *copy f to g* | *read g, write f* |
|---|---|
| `x = p.f` | `y = p.g` |
| `p.g = x` | `p.f = 1` |
| `return x` | `return y` |

- In SC semantics, `return 1` impossible
- Can result from non-conflict reordering

| | | |
|---|---|---|
| Memory: | `p.f=1` | `p.g=1` |
| Threads: | `1 = p.f`<br>`p.g = 1`<br>`return 1` | `p.f = 1`<br>`1 = p.g`<br>`return 1` |

# Thin-air read: making things up

- Need to be careful
- Program B:

```
copy f to g      copy g to f
x = p.f          y = p.g
p.g = x          p.f = y
return x         return y
```

- `return 1` undesirable — out of “thin air”

Memory:

```
Threads:   x = p.f          y = p.g
           p.g = x          p.f = y
           return x         return y
```

# Thin-air read: making things up

- Need to be careful
- Program B:

```
copy f to g        copy g to f
x = p.f            y = p.g
p.g = x            p.f = y
return x           return y
```

- `return 1` undesirable — out of “thin air”

```
Memory:    p.f=0        p.g=0
Threads:   x = p.f      y = p.g
           p.g = x      p.f = y
           return x     return y
```

# Thin-air read: making things up

- Need to be careful
- Program B:

```
copy f to g       copy g to f
x = p.f           y = p.g
p.g = x           p.f = y
return x          return y
```

- `return 1` undesirable — out of “thin air”

```
Memory:    p.f=0          p.g=0

Threads:   x = p.f        0 = p.g
           p.g = x        p.f = 0
           return x       return 0
```

# Thin-air read: making things up

- Need to be careful
- Program B:

```
copy f to g     copy g to f
x = p.f         y = p.g
p.g = x         p.f = y
return x        return y
```

- `return 1` undesirable — out of “thin air”

```
Memory:    p.f=0        p.g=0

Threads:   x = p.f       0 = p.g
           p.g = x       p.f = 0
           return x      return 0
```

# Thin-air read: making things up

- Need to be careful
- Program B:

```
copy f to g      copy g to f
x = p.f          y = p.g
p.g = x          p.f = y
return x         return y
```

- `return 1` undesirable — out of “thin air”

```
Memory:    p.f=0        p.g=0

Threads:   0 = p.f      0 = p.g
           p.g = 0      p.f = 0
           return 0     return 0
```

# Thin-air read: making things up

- Need to be careful
- Program B:

```
copy f to g      copy g to f
x = p.f          y = p.g
p.g = x          p.f = y
return x         return y
```

- `return 1` undesirable — out of “thin air”

```
Memory:    p.f=0          p.g=0

Threads:   0 = p.f        0 = p.g
           p.g = 0        p.f = 0
           return 0       return 0
```

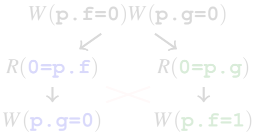
# Java Memory Model (JMM)

- Program A:

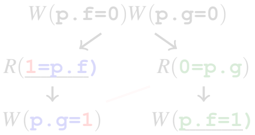
```
copy f to g       read g, write f
x = p.f           y = p.g
p.g = x           p.f = 1
return x          return y
```

- JMM allows both threads to return 1
- Critism of JMM:
  - Language acceptor, not generator
  - Difficult to understand (still)
  - Invalidates useful optimizations (still)
    - Redundant read elimination
    - Roach motel

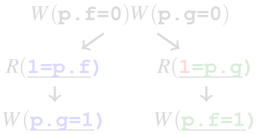
Execution 1:



Execution 2:



Execution 3:



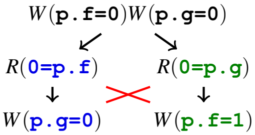
# Java Memory Model (JMM)

- Program A:

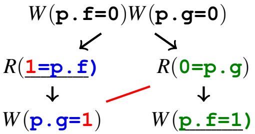
```
copy f to g        read g, write f
x = p.f            y = p.g
p.g = x            p.f = 1
return x           return y
```

- JMM allows both threads to return 1
- Critism of JMM:
  - Language acceptor, not generator
  - Difficult to understand (still)
  - Invalidates useful optimizations (still)
    - Redundant read elimination
    - Roach motel

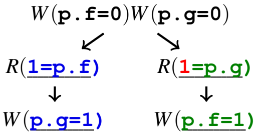
Execution 1:



Execution 2:



Execution 3:



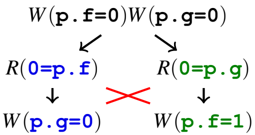
# Java Memory Model (JMM)

- Program A:

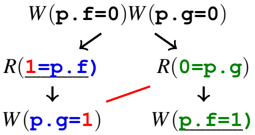
```
copy f to g       read g, write f
x = p.f           y = p.g
p.g = x           p.f = 1
return x          return y
```

- JMM allows both threads to return 1
- Critism of JMM:
  - Language acceptor, not generator
  - Difficult to understand (still)
  - Invalidates useful optimizations (still)
    - Redundant read elimination
    - Roach motel

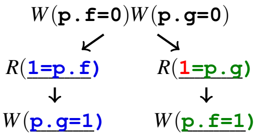
Execution 1:



Execution 2:



Execution 3:



# Our goals

- New formalization of JMM
- Generative model
- Standard guarantees
  - DRF — DRF programs have SC executions
  - no TAR — no "Thin Air Reads"
- Strictly more expressive than JMM
  - Every outcome allowed by JMM allowed by our semantics
  - Only for lockless programs

# Our goals

- New formalization of JMM
- Generative model
- Standard guarantees
  - DRF — DRF programs have SC executions
  - no TAR — no “Thin Air Reads”
- Strictly more expressive than JMM
  - Every outcome allowed by JMM allowed by our semantics
  - Only for lockless programs

# Our goals

- New formalization of JMM
- Generative model
- Standard guarantees
    - DRF — DRF programs have SC executions
    - no TAR — no “Thin Air Reads”
- Strictly more expressive than JMM
    - Every outcome allowed by JMM allowed by our semantics
    - Only for lockless programs

# Related Work

- *The Java Memory Model*
  Manson (PhD Thesis 2004)
  Also Manson, Pugh, Adve, (POPL 2005)
- *Foundations of the C++ Concurrency Memory Model*
  Boehm and Adve (PLDI 2008)
- *Program Transformations in Weak Memory Models*
  Sevcík (PhD Thesis, 2008)
  Also Sevcík and Aspinall (ECOOP 2008)
- *The semantics of x86-CC multiprocessor machine code*
  Sarkar, Sewell, Nardelli, Owens, Ridge, Braibant, Myreen, Alglave (POPL 2009)
- *Relaxed memory models: an operational approach*
  Boudol and Petri (POPL 2009)
  Also Boudol and Petri (ESOP 2010)

1. Background
   - Sequential Consistency
   - Data Race Free Model
   - Java Memory Model
2. Speculative semantics
   - Empirical and speculative actions
   - Desirable executions allowed
   - Undesirable executions prevented
3. Summary of results
   - Relation to Java Memory Model
   - Simulation precongruence

# Memory as action sequence

- No conventional memory
- Write action: **`!p.f=1`**  
  Aquire action: $\curvearrowright$**`k`**  
  Release action: **`k`**$\curvearrowleft$
- Read value determined by context (Boudol and Petri, 2009)  
  Context is a sequence of actions

| | | | |
|---|---|---|---|
| $\curvearrowright$**`k`** | **`!p.f=1`** | **`!p.g=1`** | **`k`**$\curvearrowleft$ |
| **`!p.f=2`** | $\curvearrowright$**`k`** | **`!p.g=2`** | [ ] |

- **Visibility** standard from JMM `p.f` visible at `1` and `2`  
  `p.g` visible at `2` only, due to **`k`**
- Threads may **reorder** non-conflict actions privately (see paper)

# Memory as action sequence

- No conventional memory
- Write action: **!p.f=1**
  Aquire action: $\curvearrowright$**k**
  Release action: **k**$\curvearrowleft$
- Read value determined by context (Boudol and Petri, 2009)
  Context is a sequence of actions

| | | | |
|---|---|---|---|
| $\curvearrowright$**k** | **!p.f=1** | **!p.g=1** | **k**$\curvearrowleft$ |
| **!p.f=2** | $\curvearrowright$**k** | **!p.g=2** | [ ] |

- **Visibility** standard from JMM **p.f** visible at **1** and **2**
  **p.g** visible at **2** only, due to **k**
- Threads may **reorder** non-conflict actions privately (see paper)

# Memory as action sequence

- No conventional memory
- Write action: `!p.f=1`
  Aquire action: ↷**k**
  Release action: **k**↶
- Read value determined by context (Boudol and Petri, 2009)
  Context is a sequence of actions

| | | | |
|---|---|---|---|
| ↷k | !p.f=1 | ~~!p.g=1~~ | k↶ |
| !p.f=2 | ↷k | !p.g=2 | [ ] |

- **Visibility** standard from JMM `p.f` visible at 1 and 2
  `p.g` visible at 2 only, due to `k`
- Threads may **reorder** non-conflict actions privately (see paper)

# Memory as action sequence

- No conventional memory
- Write action: `!p.f=1`
  Aquire action: $\curvearrowright$**`k`**
  Release action: **`k`**$\curvearrowleft$
- Read value determined by context (Boudol and Petri, 2009)
  Context is a sequence of actions

| | | | |
|---|---|---|---|
| $\curvearrowright$**`k`** | **`!p.f=1`** | **`!p.g=1`** | **`k`**$\curvearrowleft$ |
| **`!p.f=2`** | $\curvearrowright$**`k`** | **`!p.g=2`** | [ ] |

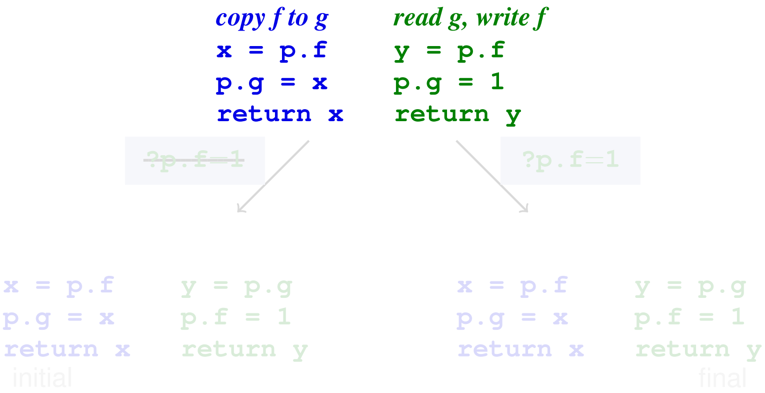
- **Visibility** standard from JMM **`p.f`** visible at **`1`** and **`2`**
  **`p.g`** visible at **`2`** only, due to **`k`**
- Threads may **reorder** non-conflict actions privately (see paper)

# Speculative action

- Speculation **`?p.f=1`** causes branching
- Worlds execute independently: **initial** and **final**
  - Speculation visible in final branch, not initial
  - Initial branch must produce justifying empirical write **`!p.f=1`**

# Initiality not too restrictive: Program A

- Initial branch must **justify** speculation
- Afterwards, only final copy remains



*copy f to g*

```
x = p.f
p.g = x
return x
```

*read g, write f*

```
y = p.f
p.g = 1
return y
```

~~?p.f=1~~

?p.f=1

```
x = p.f        y = p.g
p.g = x        p.f = 1
return x       return y
```

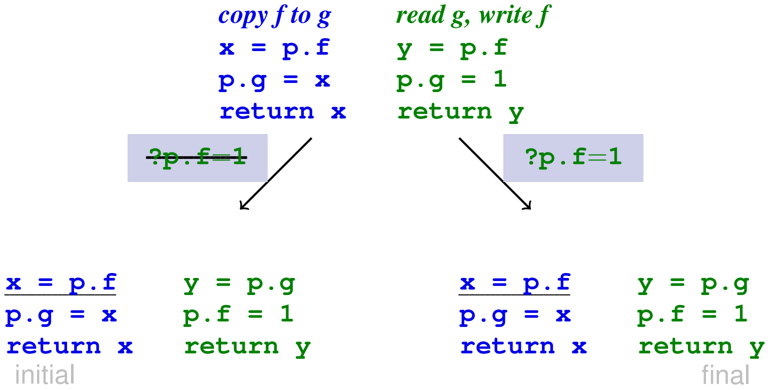
initial

```
x = p.f        y = p.g
p.g = x        p.f = 1
return x       return y
```

final

# Initiality not too restrictive: Program A

- Initial branch must **justify** speculation
- Afterwards, only final copy remains



*copy f to g* *read g, write f*

```
x = p.f        y = p.f
p.g = x        p.g = 1
return x       return y
```

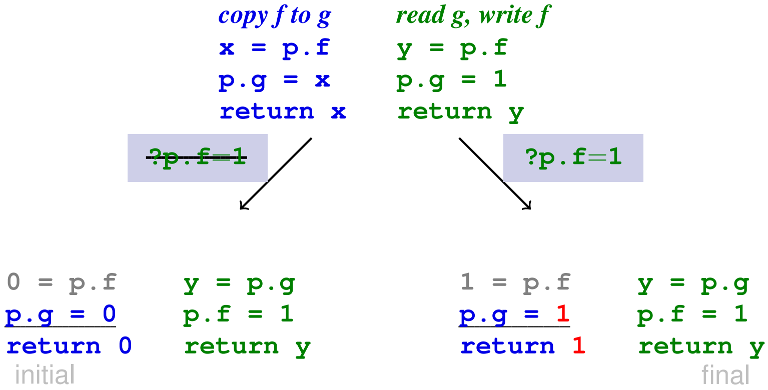
~~?p.f=1~~

?p.f=1

```
x = p.f        y = p.g        x = p.f        y = p.g
p.g = x        p.f = 1        p.g = x        p.f = 1
return x       return y       return x       return y
 initial                                          final
```

# Initiality not too restrictive: Program A

- Initial branch must **justify** speculation
- Afterwards, only final copy remains



*copy f to g*

```
x = p.f
p.g = x
return x
```

*read g, write f*

```
y = p.f
p.g = 1
return y
```

~~?p.f=1~~

?p.f=1

```
0 = p.f      y = p.g
p.g = 0      p.f = 1
return 0     return y
```

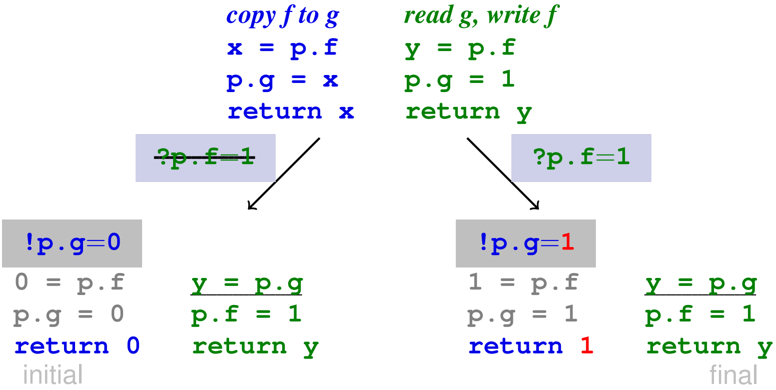
initial

```
1 = p.f      y = p.g
p.g = 1      p.f = 1
return 1     return y
```

final

# Initiality not too restrictive: Program A

- Initial branch must **justify** speculation
- Afterwards, only final copy remains



*copy f to g*

```
x = p.f
p.g = x
return x
```

*read g, write f*

```
y = p.f
p.g = 1
return y
```

~~?p.f=1~~

?p.f=1

!p.g=0

```
0 = p.f
p.g = 0
return 0
```

initial

```
y = p.g
p.f = 1
return y
```

!p.g=1

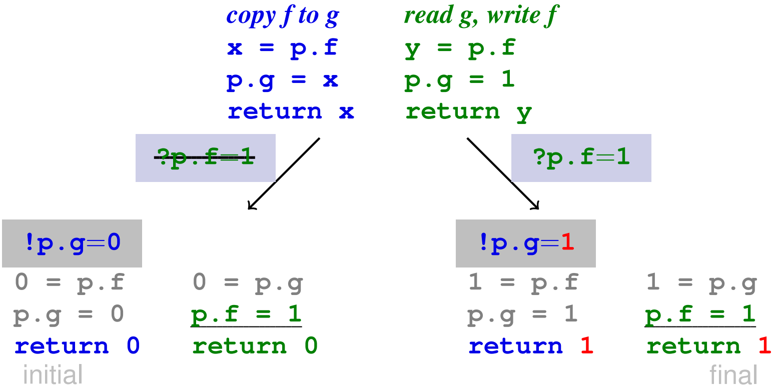
```
1 = p.f
p.g = 1
return 1
```

```
y = p.g
p.f = 1
return y
```

final

# Initiality not too restrictive: Program A

- Initial branch must **justify** speculation
- Afterwards, only final copy remains



*copy f to g*

```
x = p.f
p.g = x
return x
```

*read g, write f*

```
y = p.f
p.g = 1
return y
```

~~?p.f=1~~

?p.f=1

!p.g=0

```
0 = p.f
p.g = 0
return 0
```

initial

```
0 = p.g
p.f = 1
return 0
```

!p.g=1

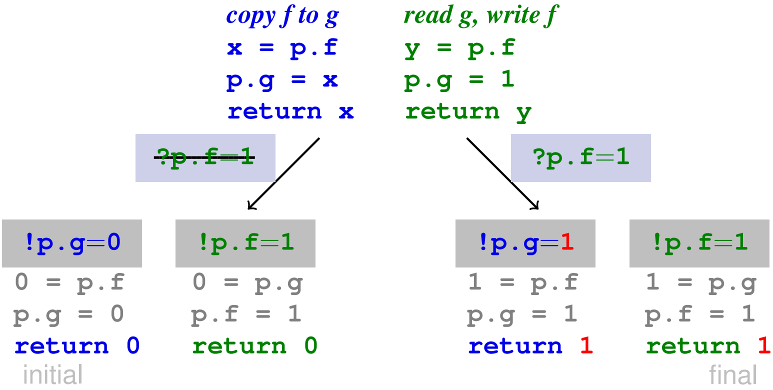
```
1 = p.f
p.g = 1
return 1
```

```
1 = p.g
p.f = 1
return 1
```

final

# Initiality not too restrictive: Program A

- Initial branch must **justify** speculation
- Afterwards, only final copy remains



*copy f to g*

```
x = p.f
p.g = x
return x
```

*read g, write f*

```
y = p.f
p.g = 1
return y
```

~~?p.f=1~~

?p.f=1

!p.g=0

```
0 = p.f
p.g = 0
return 0
```

initial

!p.f=1

```
0 = p.g
p.f = 1
return 0
```

!p.g=1

```
1 = p.f
p.g = 1
return 1
```

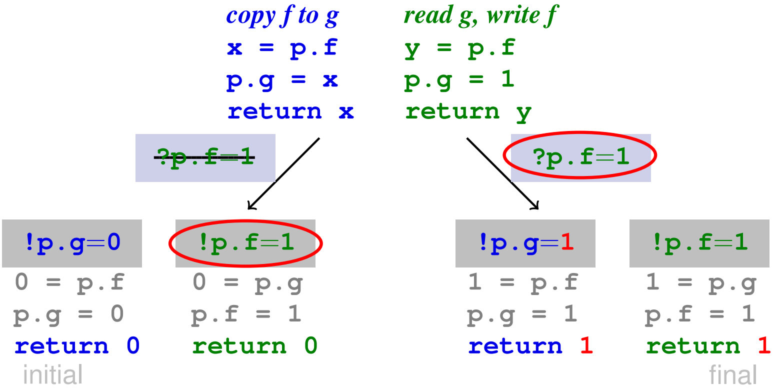
!p.f=1

```
1 = p.g
p.f = 1
return 1
```

final

# Initiality not too restrictive: Program A

- Initial branch must **justify** speculation
- Afterwards, only final copy remains



*copy f to g*

```
x = p.f
p.g = x
return x
```

*read g, write f*

```
y = p.f
p.g = 1
return y
```

~~?p.f=1~~

?p.f=1

!p.g=0

```
0 = p.f
p.g = 0
return 0
```

initial

!p.f=1

```
0 = p.g
p.f = 1
return 0
```

!p.g=1

```
1 = p.f
p.g = 1
return 1
```

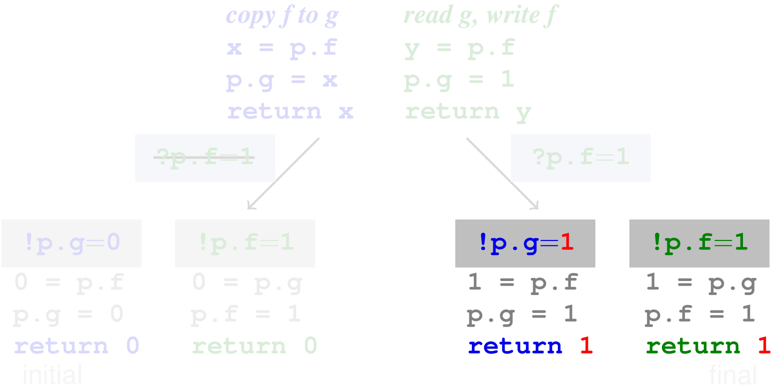
!p.f=1

```
1 = p.g
p.f = 1
return 1
```

final

# Initiality not too restrictive: Program A

- Initial branch must **justify** speculation
- Afterwards, only final copy remains



*copy f to g*

```
x = p.f
p.g = x
return x
```

*read g, write f*

```
y = p.f
p.g = 1
return y
```

~~?p.f=1~~

?p.f=1

```
!p.g=0
0 = p.f
p.g = 0
return 0
```

initial

```
!p.f=1
0 = p.g
p.f = 1
return 0
```

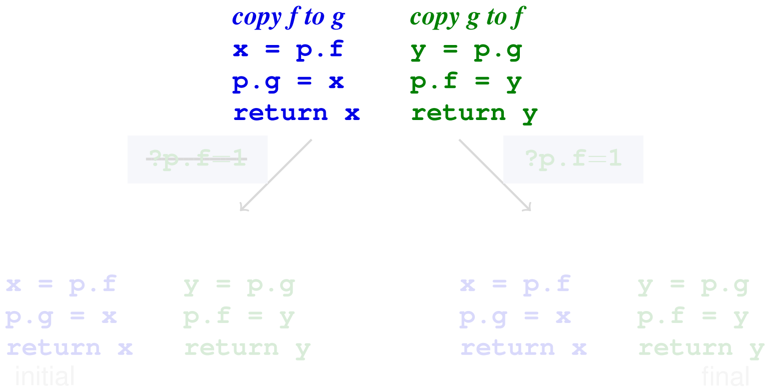
```
!p.g=1
1 = p.f
p.g = 1
return 1
```

```
!p.f=1
1 = p.g
p.f = 1
return 1
```

final

# Initiality necessary: Program B

- Initial branch must **justify** speculation
- Otherwise, execution is stuck



```
copy f to g      copy g to f
x = p.f          y = p.g
p.g = x          p.f = y
return x         return y
```

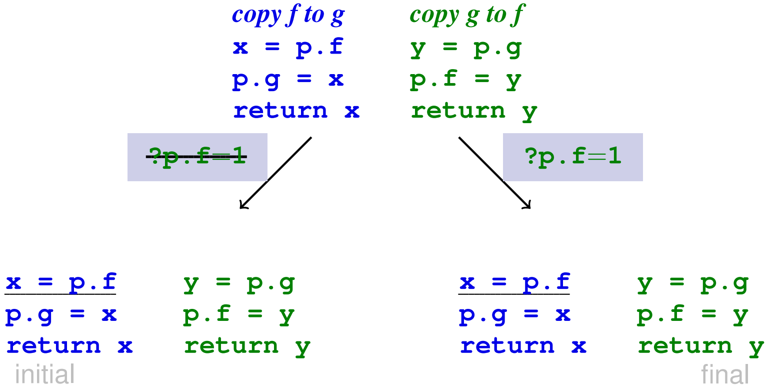
~~?p.f=1~~

?p.f=1

```
x = p.f     y = p.g          x = p.f     y = p.g
p.g = x     p.f = y          p.g = x     p.f = y
return x    return y         return x    return y
 initial                                        final
```

# Initiality necessary: Program B

- Initial branch must **justify** speculation
- Otherwise, execution is stuck



*copy f to g*　　*copy g to f*

```
x = p.f          y = p.g
p.g = x          p.f = y
return x         return y
```

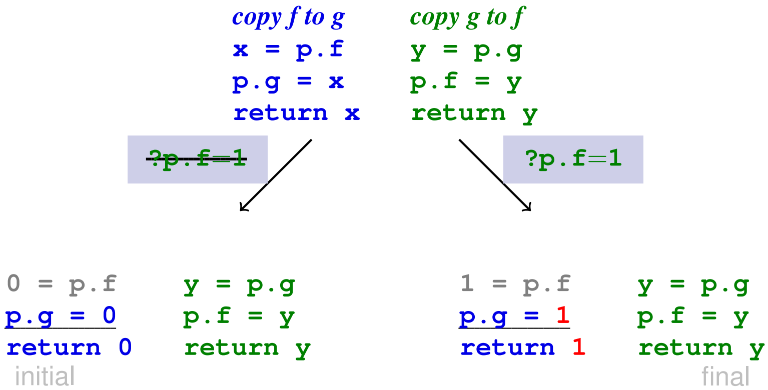
~~?p.f=1~~

?p.f=1

```
x = p.f      y = p.g          x = p.f      y = p.g
p.g = x      p.f = y          p.g = x      p.f = y
return x     return y         return x     return y
```

# Initiality necessary: Program B

- Initial branch must **justify** speculation
- Otherwise, execution is stuck



*copy f to g* *copy g to f*

```
x = p.f       y = p.g
p.g = x       p.f = y
return x      return y
```

~~?p.f=1~~

?p.f=1

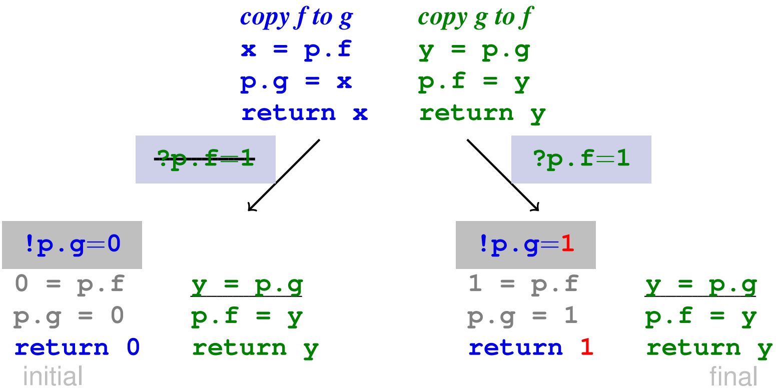
```
0 = p.f       y = p.g          1 = p.f       y = p.g
p.g = 0       p.f = y          p.g = 1       p.f = y
return 0      return y         return 1      return y
```

initial

final

# Initiality necessary: Program B

- Initial branch must **justify** speculation
- Otherwise, execution is stuck



*copy f to g*

```
x = p.f
p.g = x
return x
```

*copy g to f*

```
y = p.g
p.f = y
return y
```

~~?p.f=1~~

?p.f=1

!p.g=0

```
0 = p.f
p.g = 0
return 0
```

initial

```
y = p.g
p.f = y
return y
```

!p.g=1

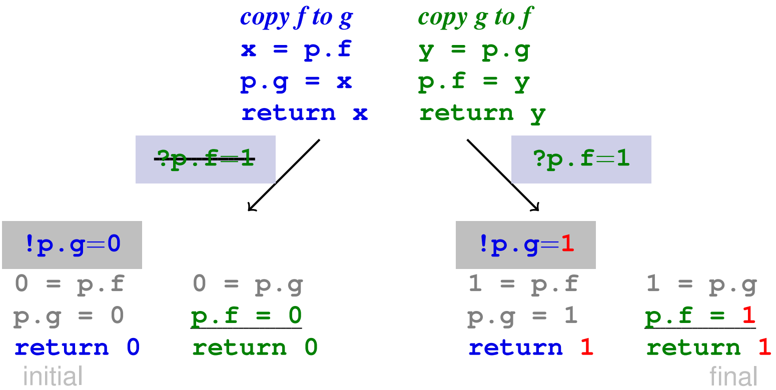
```
1 = p.f
p.g = 1
return 1
```

```
y = p.g
p.f = y
return y
```

final

# Initiality necessary: Program B

- Initial branch must **justify** speculation
- Otherwise, execution is stuck



*copy f to g*

```
x = p.f
p.g = x
return x
```

*copy g to f*

```
y = p.g
p.f = y
return y
```

~~?p.f=1~~

?p.f=1

!p.g=0

```
0 = p.f
p.g = 0
return 0
```

initial

```
0 = p.g
p.f = 0
return 0
```

!p.g=1

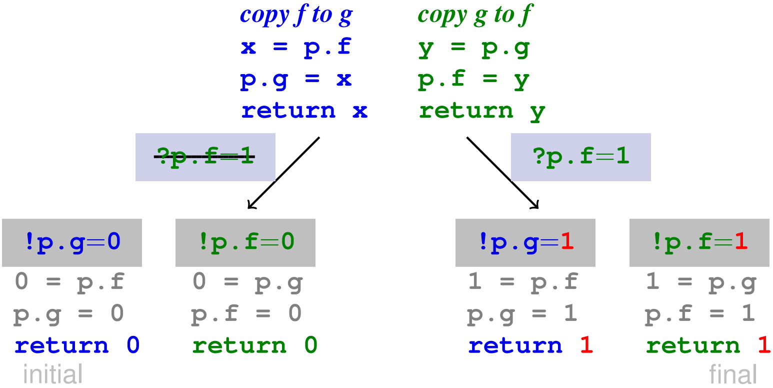
```
1 = p.f
p.g = 1
return 1
```

```
1 = p.g
p.f = 1
return 1
```

final

# Initiality necessary: Program B

- Initial branch must **justify** speculation
- Otherwise, execution is stuck



*copy f to g*

```
x = p.f
p.g = x
return x
```

*copy g to f*

```
y = p.g
p.f = y
return y
```

~~?p.f=1~~

?p.f=1

```
!p.g=0
0 = p.f
p.g = 0
return 0
```

initial

```
!p.f=0
0 = p.g
p.f = 0
return 0
```

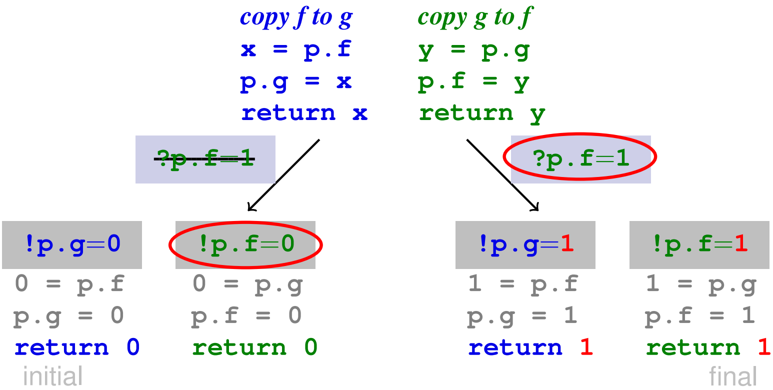
```
!p.g=1
1 = p.f
p.g = 1
return 1
```

```
!p.f=1
1 = p.g
p.f = 1
return 1
```

final

# Initiality necessary: Program B

- Initial branch must **justify** speculation
- Otherwise, execution is stuck



*copy f to g*

```
x = p.f
p.g = x
return x
```

*copy g to f*

```
y = p.g
p.f = y
return y
```

~~?p.f=1~~

?p.f=1

```
!p.g=0
0 = p.f
p.g = 0
return 0
```

initial

```
!p.f=0
0 = p.g
p.f = 0
return 0
```

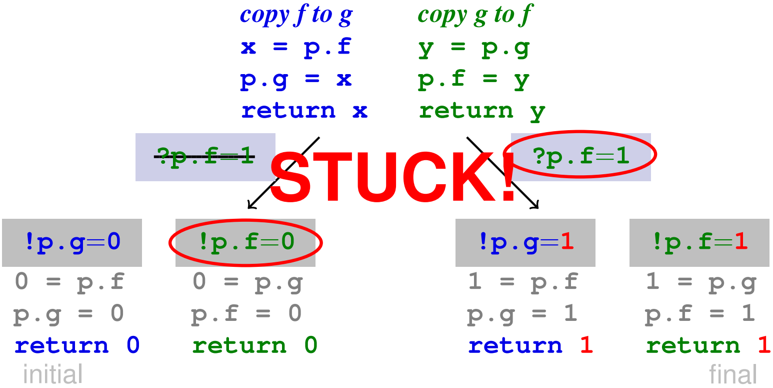
```
!p.g=1
1 = p.f
p.g = 1
return 1
```

```
!p.f=1
1 = p.g
p.f = 1
return 1
```

final

# Initiality necessary: Program B

- Initial branch must **justify** speculation
- Otherwise, execution is stuck



*copy f to g*

```
x = p.f
p.g = x
return x
```

*copy g to f*

```
y = p.g
p.f = y
return y
```

~~?p.f=1~~

**STUCK!**

?p.f=1

!p.g=0

```
0 = p.f
p.g = 0
return 0
```

initial

!p.f=0

```
0 = p.g
p.f = 0
return 0
```

!p.g=1

```
1 = p.f
p.g = 1
return 1
```

!p.f=1

```
1 = p.g
p.f = 1
return 1
```

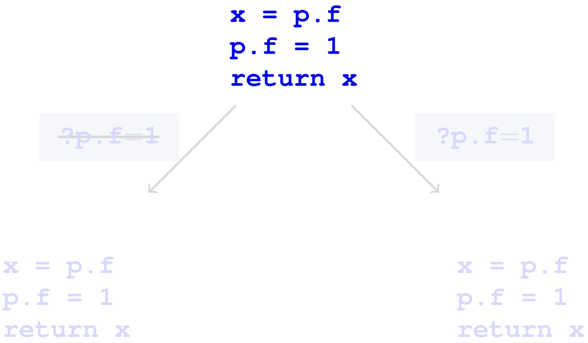
final

# Avoiding thin air reads

- ✓ Initiality
- ? No self-justification
- ? Consistency
- ? Timeliness

# Self justification: a degenerate case

- Impossible in SC execution: `return 1`
- This execution prevented by definition of visiblity:
  Thread can not see self-speculation



```
x = p.f
p.f = 1
return x
```

~~?p.f=1~~

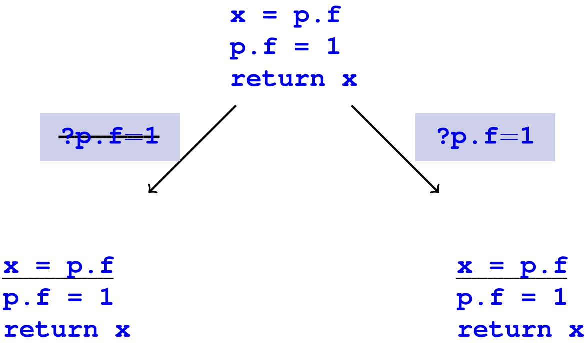
?p.f=1

```
x = p.f
p.f = 1
return x
```

```
x = p.f
p.f = 1
return x
```

# Self justification: a degenerate case

- Impossible in SC execution: `return 1`
- This execution prevented by definition of visiblity: Thread can not see self-speculation



```
x = p.f
p.f = 1
return x
```

~~?p.f=1~~

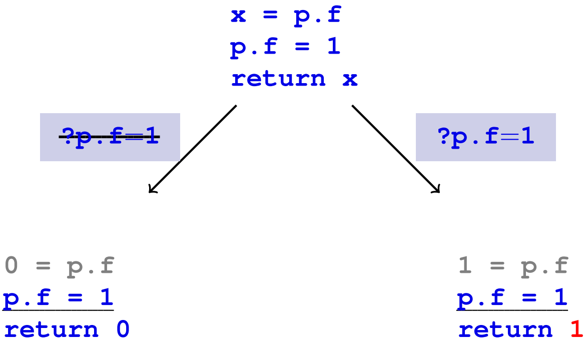
?p.f=1

```
x = p.f
p.f = 1
return x
```

```
x = p.f
p.f = 1
return x
```

# Self justification: a degenerate case

- Impossible in SC execution: `return 1`
- This execution prevented by definition of visiblity: Thread can not see self-speculation



```
x = p.f
p.f = 1
return x
```

~~?p.f=1~~

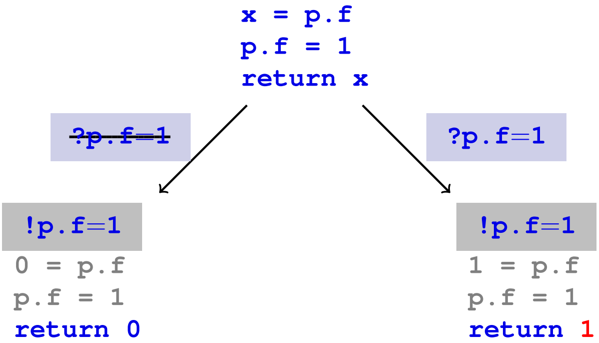
?p.f=1

```
0 = p.f
p.f = 1
return 0
```

```
1 = p.f
p.f = 1
return 1
```

# Self justification: a degenerate case

- Impossible in SC execution: `return 1`
- This execution prevented by definition of visiblity: Thread can not see self-speculation



```
x = p.f
p.f = 1
return x
```

~~?p.f=1~~

?p.f=1

!p.f=1

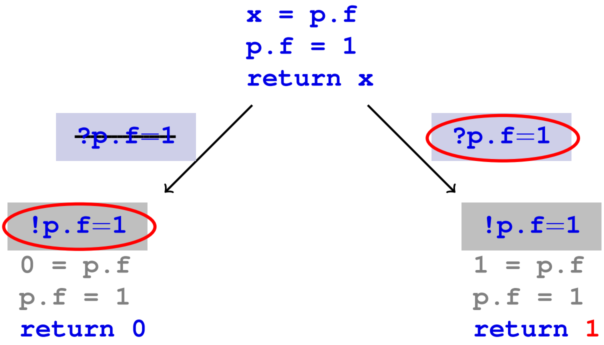
```
0 = p.f
p.f = 1
return 0
```

!p.f=1

```
1 = p.f
p.f = 1
return 1
```

# Self justification: a degenerate case

- Impossible in SC execution: `return 1`
- This execution prevented by definition of visiblity: Thread can not see self-speculation



```
x = p.f
p.f = 1
return x
```

~~?p.f=1~~

!p.f=1

```
0 = p.f
p.f = 1
return 0
```

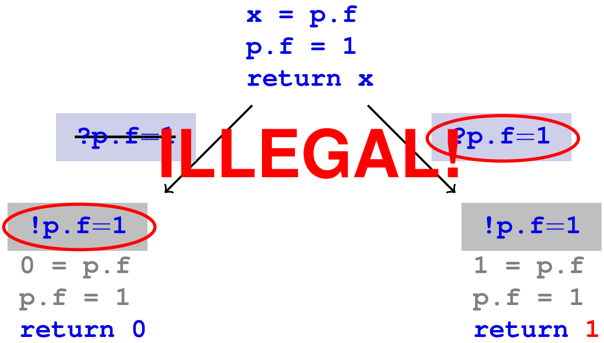
?p.f=1

!p.f=1

```
1 = p.f
p.f = 1
return 1
```

# Self justification: a degenerate case

- Impossible in SC execution: `return 1`
- This execution prevented by definition of visiblity: Thread can not see self-speculation



```
x = p.f
p.f = 1
return x
```

~~?p.f=1~~

ILLEGAL!

?p.f=1

!p.f=1

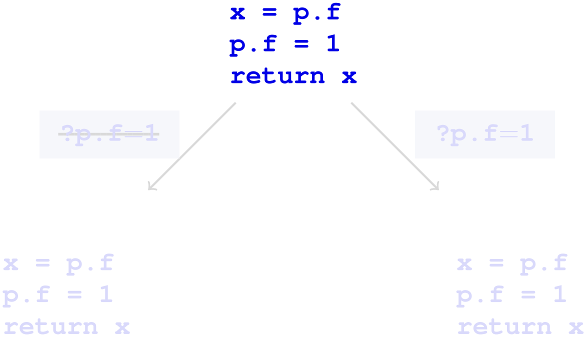
```
0 = p.f
p.f = 1
return 0
```

!p.f=1

```
1 = p.f
p.f = 1
return 1
```

# Self justification: a degenerate case

- Impossible in SC execution: `return 1`
- This execution prevented by definition of visiblity: Thread can not see self-speculation



```
x = p.f
p.f = 1
return x
```

~~?p.f=1~~

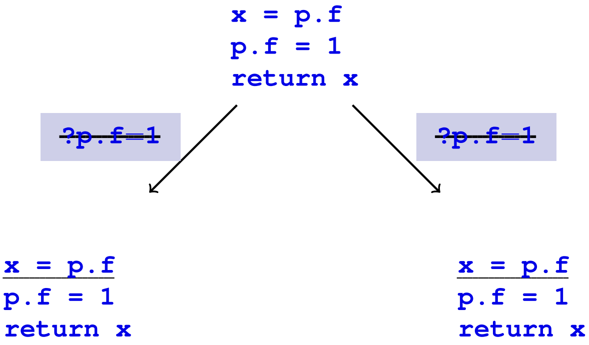
?p.f=1

```
x = p.f
p.f = 1
return x
```

```
x = p.f
p.f = 1
return x
```

# Self justification: a degenerate case

- Impossible in SC execution: `return 1`
- This execution prevented by definition of visiblity: Thread can not see self-speculation



```
x = p.f
p.f = 1
return x
```

~~?p.f=1~~

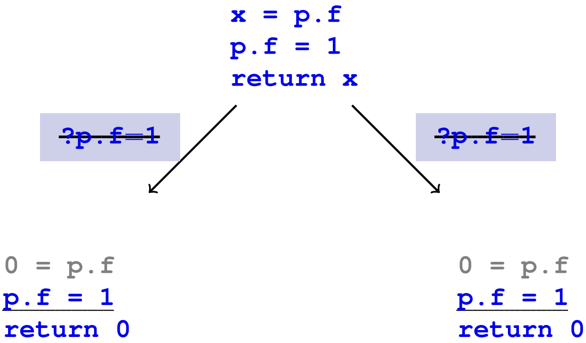
~~?p.f=1~~

```
x = p.f
p.f = 1
return x
```

```
x = p.f
p.f = 1
return x
```

# Self justification: a degenerate case

- Impossible in SC execution: `return 1`
- This execution prevented by definition of visiblity: Thread can not see self-speculation



```
x = p.f
p.f = 1
return x
```

~~?p.f=1~~

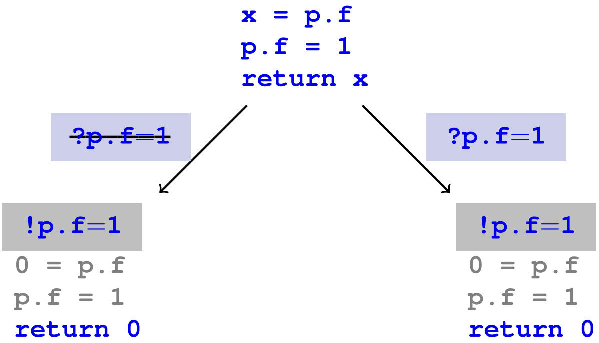
~~?p.f=1~~

```
0 = p.f
p.f = 1
return 0
```

```
0 = p.f
p.f = 1
return 0
```

# Self justification: a degenerate case

- Impossible in SC execution: `return 1`
- This execution prevented by definition of visiblity: Thread can not see self-speculation



```
x = p.f
p.f = 1
return x
```

~~?p.f=1~~

?p.f=1

!p.f=1

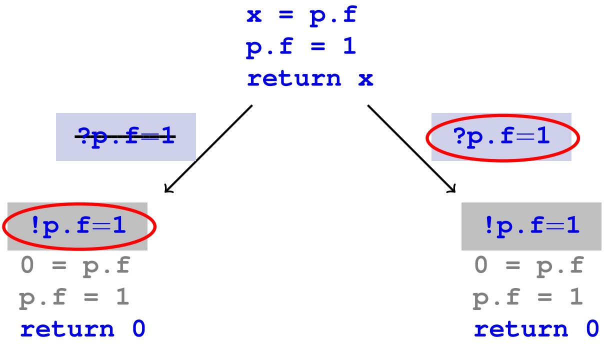
```
0 = p.f
p.f = 1
return 0
```

!p.f=1

```
0 = p.f
p.f = 1
return 0
```

# Self justification: a degenerate case

- Impossible in SC execution: `return 1`
- This execution prevented by definition of visiblity:
  Thread can not see self-speculation



```
x = p.f
p.f = 1
return x
```

~~?p.f=1~~

!p.f=1

```
0 = p.f
p.f = 1
return 0
```

?p.f=1

!p.f=1

```
0 = p.f
p.f = 1
return 0
```

# Controlling speculation: consistency

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f       x = p.f       x = p.f
  if(x==0)      if(x==0)      p.g = x       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

- Impossible in SC execution: `return(1,2)`
- Possible in final branch with speculation `?p.f=2`
- Initial branch can produce justifying write
- Inconsistent use of locks between speculation and justifying write

Initial branch

| | | |
|---|---|---|
| !p.f=0 | !p.g=0 | ~~?p.f=2~~ |
| ↷k | !p.g=0 | k↶ |
| ↷k | !p.f=2 | k↶ |
| ↷k | k↶ | |
| ↷k | k↶ | |

Final branch

| | | |
|---|---|---|
| !p.f=0 | !p.g=0 | ?p.f=2 |
| ↷k | !p.g=2 | k↶ |
| ↷k | !p.f=1 | k↶ |
| ↷k | k↶ | |
| ↷k | k↶ | |

# Controlling speculation: consistency

```
k.acquire()      k.acquire()      k.acquire()      k.acquire()
  x = p.f          x = p.f          x = p.f          x = p.f
  if(x==0)         if(x==0)         p.g = x          y = p.g
    p.f = 1          p.f = 2      k.release()      k.release()
k.release()      k.release()                       return(x,y)
```

- Impossible in SC execution: `return(1,2)`
- Possible in final branch with speculation `?p.f=2`
- Initial branch can produce justifying write
- Inconsistent use of locks between speculation and justifying write

Initial branch

| | | |
|---|---|---|
| !p.f=0 | !p.g=0 | ~~?p.f=2~~ |
| ↷k | !p.g=0 | k↶ |
| ↷k | !p.f=2 | k↶ |
| ↷k | k↶ | |
| ↷k | k↶ | |

Final branch

| | | |
|---|---|---|
| !p.f=0 | !p.g=0 | ?p.f=2 |
| ↷k | !p.g=2 | k↶ |
| ↷k | !p.f=1 | k↶ |
| ↷k | k↶ | |
| ↷k | k↶ | |

# Controlling speculation: consistency

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f       x = p.f       x = p.f
  if(x==0)      if(x==0)      p.g = x       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

- Impossible in SC execution: `return(1,2)`
- Possible in final branch with speculation `?p.f=2`
- Initial branch can produce justifying write
- Inconsistent use of locks between speculation and justifying write

Initial branch

| | | |
|---|---|---|
| !p.f=0 | !p.g=0 | ~~?p.f=2~~ |
| ↷k | !p.g=0 | k↶ |
| ↷k | !p.f=2 | k↶ |
| ↷k | k↶ | |
| ↷k | k↶ | |

Final branch

| | | |
|---|---|---|
| !p.f=0 | !p.g=0 | ?p.f=2 |
| ↷k | !p.g=2 | k↶ |
| ↷k | !p.f=1 | k↶ |
| ↷k | k↶ | |
| ↷k | k↶ | |

# Controlling speculation: consistency

```
k.acquire()    k.acquire()    k.acquire()    k.acquire()
  x = p.f        x = p.f        x = p.f        x = p.f
  if(x==0)       if(x==0)       p.g = x        y = p.g
    p.f = 1        p.f = 2    k.release()    k.release()
k.release()    k.release()                   return(x,y)
```

- Impossible in SC execution: `return(1,2)`
- Possible in final branch with speculation `?p.f=2`
- Initial branch can produce justifying write
- Inconsistent use of locks between speculation and justifying write

Initial branch

| | | |
|---|---|---|
| !p.f=0 | !p.g=0 | ~~?p.f=2~~ |
| ↷k | !p.g=0 | k↶ |
| ↷k | !p.f=2 | k↶ |
| ↷k | k↶ | |
| ↷k | k↶ | |

STUCK!

Final branch

| | | |
|---|---|---|
| !p.f=0 | !p.g=0 | ?p.f=2 |
| ↷k | !p.g=2 | k↶ |
| ↷k | !p.f=1 | k↶ |
| ↷k | k↶ | |
| ↷k | k↶ | |

# Controlling speculation: timeliness

```
k.acquire()  k.acquire()  k.acquire()
  x = p.f      x = p.f
  p.f = x+1    p.f = x+1
  p.g = 1      p.g = 2      y = p.g
k.release()  k.release()  k.release()
return x     return x     return y
```

- Impossible SC: `return 0; return 1; return 1;`
- Possible in final branch with speculation `?p.g=1`
- Initial and final branches produce same actions
- Speculation used to introduce data race in final branch

| | | | | |
|---|---|---|---|---|
| `!p.f=0` | `!p.g=0` | | | |
| ↷`k` | `!p.f=1` | `!p.g=1` | `k`↶ | ~~`?p.g=1`~~ |
| ↷`k` | `!p.f=2` | `!p.g=2` | `k`↶ | |
| ↷`k` | `k`↶ | | | |

# Controlling speculation: timeliness

```
k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f
  p.f = x+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return x      return x      return y
```

- Impossible SC: `return 0; return 1; return 1;`
- Possible in final branch with speculation `?p.g=1`
- Initial and final branches produce same actions
- Speculation used to introduce data race in final branch

| | | | | |
|---|---|---|---|---|
| !p.f=0 | !p.g=0 | | | |
| ↷k | !p.f=1 | !p.g=1 | k↶ | ~~?p.g=1~~ |
| ↷k | !p.f=2 | !p.g=2 | k↶ | |
| ↷k | k↶ | | | |

# Controlling speculation: timeliness

```
k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f
  p.f = x+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return x      return x      return y
```

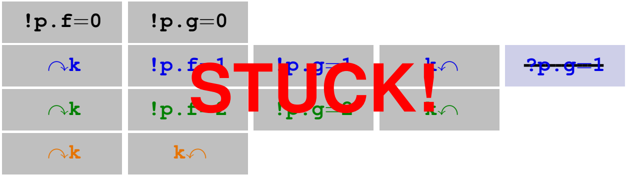
- Impossible SC: `return 0;` `return 1;` `return 1;`
- Possible in final branch with speculation `?p.g=1`
- Initial and final branches produce same actions
- Speculation used to introduce data race in final branch

| | | | | |
|---|---|---|---|---|
| !p.f=0 | !p.g=0 | | | |
| ↷k | !p.f=1 | !p.g=1 | k↶ | ~~?p.g=1~~ |
| ↷k | !p.f=2 | !p.g=2 | k↶ | |
| ↷k | k↶ | | | |

# Controlling speculation: timeliness

```
k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f
  p.f = x+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return x      return x      return y
```

- Impossible SC: `return 0; return 1; return 1;`
- Possible in final branch with speculation `?p.g=1`
- Initial and final branches produce same actions
- Speculation used to introduce data race in final branch



1 Background
- Sequential Consistency
- Data Race Free Model
- Java Memory Model

2 Speculative semantics
- Empirical and speculative actions
- Desirable executions allowed
- Undesirable executions prevented

3 Summary of results
- Relation to Java Memory Model
- Simulation precongruence

# Relation to JMM

Theorem

*DRF program ⇒ SC execution*

Theorem

*Lockless program ⇒ every JMM execution allowed*

# Simulation

- Simulation defined in paper
- Precongruence
- Useful

$$\texttt{p.f=1; p.g=1;} \gtrsim \texttt{p.g=1; p.f=1;}$$

$$\texttt{p.f=1; k.acquire();} \gtrsim \texttt{k.acquire(); p.f=1;}$$

$$\texttt{x=p.f; y=p.f;}\ M \gtrsim \texttt{x=p.f;}\ M\{{}^{\texttt{x}}/_{\texttt{y}}\}$$

# Summary

- New model based on **speculation**

| Data Races | Locks | New vs. JMM |
|---|---|---|
| X | - | $=$ |
| - | X | $>$ |
| ✓ | ✓ | $\not\gtrless$ |

- Simulation precongruence
- Better behaved:
  Validates redundant-read-elimination, roach-motel, etc
- Thank you

# Summary

- New model based on **speculation**

| Data Races | Locks | New vs. JMM |
|---|---|---|
| X | - | $=$ |
| - | X | $>$ |
| ✓ | ✓ | $\not\gtrless$ |

- Simulation precongruence
- Better behaved:
  Validates redundant-read-elimination, roach-motel, etc
- Thank you

# Appendix

The rest of the slides animate the execution of a few examples.

# Controlling speculation: consistency

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f       x = p.f       x = p.f
  if(x==0)      if(x==0)      p.g = x       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f       x = p.f       x = p.f
  if(x==0)      if(x==0)      p.g = x       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

# Controlling speculation: consistency

**!p.f=0** **!p.g=0** ~~?p.f=2~~

```
k.acquire()    k.acquire()    k.acquire()    k.acquire()
  x = p.f        x = p.f        x = p.f        x = p.f
  if(x==0)       if(x==0)       p.g = x        y = p.g
    p.f = 1        p.f = 2    k.release()    k.release()
k.release()    k.release()                   return(x,y)
```

**!p.f=0** **!p.g=0** ?p.f=2

```
k.acquire()    k.acquire()    k.acquire()    k.acquire()
  x = p.f        x = p.f        x = p.f        x = p.f
  if(x==0)       if(x==0)       p.g = x        y = p.g
    p.f = 1        p.f = 2    k.release()    k.release()
k.release()    k.release()                   return(x,y)
```

# Controlling speculation: consistency

| | | | |
|---|---|---|---|
| !p.f=0 | !p.g=0 | ~~?p.f=2~~ | ↷k |

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f       x = p.f       x = p.f
  if(x==0)      if(x==0)      p.g = x       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

| | | | |
|---|---|---|---|
| !p.f=0 | !p.g=0 | ?p.f=2 | ↷k |

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f       x = p.f       x = p.f
  if(x==0)      if(x==0)      p.g = x       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

# Controlling speculation: consistency

| | | | |
|---|---|---|---|
| !p.f=0 | !p.g=0 | ~~?p.f=2~~ | ↷k |

```
k.acquire()    k.acquire()    k.acquire()    k.acquire()
  x = p.f        x = p.f        0 = p.f        x = p.f
  if(x==0)       if(x==0)       p.g = 0        y = p.g
    p.f = 1        p.f = 2    k.release()    k.release()
k.release()    k.release()                   return(x,y)
```

| | | | |
|---|---|---|---|
| !p.f=0 | !p.g=0 | ?p.f=2 | ↷k |

```
k.acquire()    k.acquire()    k.acquire()    k.acquire()
  x = p.f        x = p.f        2 = p.f        x = p.f
  if(x==0)       if(x==0)       p.g = 2        y = p.g
    p.f = 1        p.f = 2    k.release()    k.release()
k.release()    k.release()                   return(x,y)
```

# Controlling speculation: consistency

| | | | | |
|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ~~?p.f=2~~ | ↷k | !p.g=0 |

```
k.acquire()    k.acquire()    k.acquire()    k.acquire()
  x = p.f        x = p.f        0 = p.f        x = p.f
  if(x==0)       if(x==0)       p.g = 0        y = p.g
    p.f = 1        p.f = 2    k.release()    k.release()
k.release()    k.release()                   return(x,y)
```

| | | | | |
|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ?p.f=2 | ↷k | !p.g=2 |

```
k.acquire()    k.acquire()    k.acquire()    k.acquire()
  x = p.f        x = p.f        2 = p.f        x = p.f
  if(x==0)       if(x==0)       p.g = 2        y = p.g
    p.f = 1        p.f = 2    k.release()    k.release()
k.release()    k.release()                   return(x,y)
```

# Controlling speculation: consistency

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ~~?p.f=2~~ | ↷k | !p.g=0 | k↶ |

```
k.acquire()     k.acquire()     k.acquire()     k.acquire()
  x = p.f         x = p.f         0 = p.f         x = p.f
  if(x==0)        if(x==0)        p.g = 0         y = p.g
    p.f = 1         p.f = 2     k.release()     k.release()
k.release()     k.release()                     return(x,y)
```

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ?p.f=2 | ↷k | !p.g=2 | k↶ |

```
k.acquire()     k.acquire()     k.acquire()     k.acquire()
  x = p.f         x = p.f         2 = p.f         x = p.f
  if(x==0)        if(x==0)        p.g = 2         y = p.g
    p.f = 1         p.f = 2     k.release()     k.release()
k.release()     k.release()                     return(x,y)
```

# Controlling speculation: consistency

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | ~~!p.g=0~~ | ~~?p.f=2~~ | ↷k | !p.g=0 | k↶ |
| ↷k | | | | | |

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f       0 = p.f       x = p.f
  if(x==0)      if(x==0)      p.g = 0       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | ~~!p.g=0~~ | ?p.f=2 | ↷k | !p.g=2 | k↶ |
| ↷k | | | | | |

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f       2 = p.f       x = p.f
  if(x==0)      if(x==0)      p.g = 2       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

# Controlling speculation: consistency

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | ~~!p.g=0~~ | ~~?p.f=2~~ | ↷k | !p.g=0 | k↶ |
| ↷k | | | | | |

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  x = p.f       0 = p.f       0 = p.f       x = p.f
  if(x==0)      if(0==0)      p.g = 0       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | ~~!p.g=0~~ | ?p.f=2 | ↷k | !p.g=2 | k↶ |
| ↷k | | | | | |

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  0 = p.f       x = p.f       2 = p.f       x = p.f
  if(0==0)      if(x==0)      p.g = 2       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

# Controlling speculation: consistency

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | ~~!p.g=0~~ | ~~?p.f=2~~ | ↷k | !p.g=0 | k↶ |
| ↷k | !p.f=2 | | | | |

```
k.acquire()     k.acquire()     k.acquire()     k.acquire()
  x = p.f         0 = p.f         0 = p.f         x = p.f
  if(x==0)        if(0==0)        p.g = 0         y = p.g
    p.f = 1         p.f = 2     k.release()     k.release()
k.release()     k.release()                     return(x,y)
```

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | ~~!p.g=0~~ | ?p.f=2 | ↷k | !p.g=2 | k↶ |
| ↷k | !p.f=1 | | | | |

```
k.acquire()     k.acquire()     k.acquire()     k.acquire()
  0 = p.f         x = p.f         2 = p.f         x = p.f
  if(0==0)        if(x==0)        p.g = 2         y = p.g
    p.f = 1         p.f = 2     k.release()     k.release()
k.release()     k.release()                     return(x,y)
```

# Controlling speculation: consistency

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ~~?p.f=2~~ | ↷k | !p.g=0 | k↶ |
| ↷k | !p.f=2 | k↶ | | | |

```
k.acquire()      k.acquire()      k.acquire()      k.acquire()
  x = p.f          0 = p.f          0 = p.f          x = p.f
  if(x==0)         if(0==0)         p.g = 0          y = p.g
    p.f = 1          p.f = 2      k.release()      k.release()
k.release()      k.release()                       return(x,y)
```

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ?p.f=2 | ↷k | !p.g=2 | k↶ |
| ↷k | !p.f=1 | k↶ | | | |

```
k.acquire()      k.acquire()      k.acquire()      k.acquire()
  0 = p.f          x = p.f          2 = p.f          x = p.f
  if(0==0)         if(x==0)         p.g = 2          y = p.g
    p.f = 1          p.f = 2      k.release()      k.release()
k.release()      k.release()                       return(x,y)
```

# Controlling speculation: consistency

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ~~?p.f=2~~ | ↷k | !p.g=0 | k↶ |
| ↷k | !p.f=2 | k↶ | ↷k | | |

```
k.acquire()    k.acquire()    k.acquire()    k.acquire()
  x = p.f        0 = p.f        0 = p.f        x = p.f
  if(x==0)       if(0==0)       p.g = 0        y = p.g
    p.f = 1        p.f = 2    k.release()    k.release()
k.release()    k.release()                   return(x,y)
```

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ~~?p.f=2~~ | ↷k | !p.g=2 | k↶ |
| ↷k | !p.f=1 | k↶ | ↷k | | |

```
k.acquire()    k.acquire()    k.acquire()    k.acquire()
  0 = p.f        x = p.f        2 = p.f        x = p.f
  if(0==0)       if(x==0)       p.g = 2        y = p.g
    p.f = 1        p.f = 2    k.release()    k.release()
k.release()    k.release()                   return(x,y)
```

# Controlling speculation: consistency

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ~~?p.f=2~~ | ↷k | !p.g=0 | k↶ |
| ↷k | !p.f=2 | k↶ | ↷k | | |

```
k.acquire()     k.acquire()     k.acquire()     k.acquire()
  2 = p.f         0 = p.f         0 = p.f         x = p.f
  if(2==0)        if(0==0)        p.g = 0         y = p.g
    p.f = 1         p.f = 2     k.release()     k.release()
k.release()     k.release()                     return(x,y)
```

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ~~?p.f=2~~ | ↷k | !p.g=2 | k↶ |
| ↷k | !p.f=1 | k↶ | ↷k | | |

```
k.acquire()     k.acquire()     k.acquire()     k.acquire()
  0 = p.f         1 = p.f         2 = p.f         x = p.f
  if(0==0)        if(1==0)        p.g = 2         y = p.g
    p.f = 1         p.f = 2     k.release()     k.release()
k.release()     k.release()                     return(x,y)
```

# Controlling speculation: consistency

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ~~?p.f=2~~ | ↷k | !p.g=0 | k↶ |
| ↷k | !p.f=2 | k↶ | ↷k | k↶ | |

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  2 = p.f       0 = p.f       0 = p.f       x = p.f
  if(2==0)      if(0==0)      p.g = 0       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ?p.f=2 | ↷k | !p.g=2 | k↶ |
| ↷k | !p.f=1 | k↶ | ↷k | k↶ | |

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  0 = p.f       1 = p.f       2 = p.f       x = p.f
  if(0==0)      if(1==0)      p.g = 2       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

# Controlling speculation: consistency

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ~~?p.f=2~~ | ↷k | !p.g=0 | k↶ |
| ↷k | !p.f=2 | k↶ | ↷k | k↶ | ↷k |

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  2 = p.f       0 = p.f       0 = p.f       x = p.f
  if(2==0)      if(0==0)      p.g = 0       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ~~?p.f=2~~ | ↷k | !p.g=2 | k↶ |
| ↷k | !p.f=1 | k↶ | ↷k | k↶ | ↷k |

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  0 = p.f       1 = p.f       2 = p.f       x = p.f
  if(0==0)      if(1==0)      p.g = 2       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(x,y)
```

# Controlling speculation: consistency

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ~~?p.f=2~~ | ↷k | !p.g=0 | k↶ |
| ↷k | !p.f=2 | k↶ | ↷k | k↶ | ↷k |

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  2 = p.f       0 = p.f       0 = p.f       2 = p.f
  if(2==0)      if(0==0)      p.g = 0       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(2,y)
```

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ~~?p.f=2~~ | ↷k | !p.g=2 | k↶ |
| ↷k | !p.f=1 | k↶ | ↷k | k↶ | ↷k |

```
k.acquire()   k.acquire()   k.acquire()   k.acquire()
  0 = p.f       1 = p.f       2 = p.f       1 = p.f
  if(0==0)      if(1==0)      p.g = 2       y = p.g
    p.f = 1       p.f = 2   k.release()   k.release()
k.release()   k.release()                 return(1,y)
```

# Controlling speculation: consistency

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ~~?p.f=2~~ | ↷k | !p.g=0 | k↶ |
| ↷k | !p.f=2 | k↶ | ↷k | k↶ | ↷k |

```
k.acquire()    k.acquire()    k.acquire()    k.acquire()
  2 = p.f        0 = p.f        0 = p.f        2 = p.f
  if(2==0)       if(0==0)       p.g = 0        0 = p.g
    p.f = 1        p.f = 2    k.release()    k.release()
k.release()    k.release()                   return(2,0)
```

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ~~?p.f=2~~ | ↷k | !p.g=2 | k↶ |
| ↷k | !p.f=1 | k↶ | ↷k | k↶ | ↷k |

```
k.acquire()    k.acquire()    k.acquire()    k.acquire()
  0 = p.f        1 = p.f        2 = p.f        1 = p.f
  if(0==0)       if(1==0)       p.g = 2        2 = p.g
    p.f = 1        p.f = 2    k.release()    k.release()
k.release()    k.release()                   return(1,2)
```

# Controlling speculation: consistency

| | | | | | | |
|---|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ~~?p.f=2~~ | ↷k | !p.g=0 | k↶ | |
| ↷k | !p.f=2 | k↶ | ↷k | k↶ | ↷k | k↶ |

```
k.acquire()    k.acquire()    k.acquire()    k.acquire()
  2 = p.f        0 = p.f        0 = p.f        2 = p.f
  if(2==0)       if(0==0)       p.g = 0        0 = p.g
    p.f = 1        p.f = 2    k.release()    k.release()
k.release()    k.release()                   return(2,0)
```

| | | | | | | |
|---|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ?p.f=2 | ↷k | !p.g=2 | k↶ | |
| ↷k | !p.f=1 | k↶ | ↷k | k↶ | ↷k | k↶ |

```
k.acquire()    k.acquire()    k.acquire()    k.acquire()
  0 = p.f        1 = p.f        2 = p.f        1 = p.f
  if(0==0)       if(1==0)       p.g = 2        2 = p.g
    p.f = 1        p.f = 2    k.release()    k.release()
k.release()    k.release()                   return(1,2)
```

# Controlling speculation: timeliness

```
k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f
  p.f = x+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return x      return x      return y
```

```
k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f
  p.f = x+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return x      return x      return y
```

# Controlling speculation: timeliness

`!p.f=0` `!p.g=0`

```
k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f
  p.f = x+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return x      return x      return y
```

`!p.f=0` `!p.g=0`

```
k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f
  p.f = x+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return x      return x      return y
```

# Controlling speculation: timeliness

| !p.f=0 | !p.g=0 | ↷k |
|---|---|---|

```
k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f
  p.f = x+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return x      return x      return y
```

| !p.f=0 | !p.g=0 | ↷k |
|---|---|---|

```
k.acquire()   k.acquire()   k.acquire()
  x = p.f       x = p.f
  p.f = x+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return x      return x      return y
```

# Controlling speculation: timeliness

```
!p.f=0    !p.g=0    ↷k
```

```
k.acquire()    k.acquire()    k.acquire()
  0 = p.f        x = p.f
  p.f = 0+1      p.f = x+1
  p.g = 1        p.g = 2        y = p.g
k.release()    k.release()    k.release()
return 0       return x       return y
```

```
!p.f=0    !p.g=0    ↷k
```

```
k.acquire()    k.acquire()    k.acquire()
  0 = p.f        x = p.f
  p.f = 0+1      p.f = x+1
  p.g = 1        p.g = 2        y = p.g
k.release()    k.release()    k.release()
return 0       return x       return y
```

# Controlling speculation: timeliness

| !p.f=0 | !p.g=0 | ↷k | !p.f=1 |
|---|---|---|---|

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       x = p.f
  p.f = 0+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return x      return y
```

| !p.f=0 | !p.g=0 | ↷k | !p.f=1 |
|---|---|---|---|

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       x = p.f
  p.f = 0+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return x      return y
```

# Controlling speculation: timeliness

| | | | | |
|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ↷k | !p.f=1 | !p.g=1 |

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       x = p.f
  p.f = 0+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return x      return y
```

| | | | | |
|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ↷k | !p.f=1 | !p.g=1 |

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       x = p.f
  p.f = 0+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return x      return y
```

# Controlling speculation: timeliness

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ↷k | !p.f=1 | !p.g=1 | k↶ |

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       x = p.f
  p.f = 0+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return x      return y
```

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ↷k | !p.f=1 | !p.g=1 | k↶ |

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       x = p.f
  p.f = 0+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return x      return y
```

# Controlling speculation: timeliness

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ~~?p.g=1~~ | | | | | |

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       x = p.f
  p.f = 0+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return x      return y
```

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ?p.g=1 | | | | | |

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       x = p.f
  p.f = 0+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return x      return y
```

# Controlling speculation: timeliness

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ~~?p.g=1~~ | | | | | |

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       x = p.f
  p.f = 0+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return x      return y
```

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ?p.g=1 | | | | | |

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       x = p.f
  p.f = 0+1     p.f = x+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return x      return y
```

# Controlling speculation: timeliness

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ~~?p.g=1~~ | ↷k | | | | |

```
k.acquire()     k.acquire()     k.acquire()
  0 = p.f         x = p.f
  p.f = 0+1       p.f = x+1
  p.g = 1         p.g = 2           y = p.g
k.release()     k.release()     k.release()
return 0        return x        return y
```

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ?p.g=1 | ↷k | | | | |

```
k.acquire()     k.acquire()     k.acquire()
  0 = p.f         x = p.f
  p.f = 0+1       p.f = x+1
  p.g = 1         p.g = 2           y = p.g
k.release()     k.release()     k.release()
return 0        return x        return y
```

# Controlling speculation: timeliness

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ~~?p.g=1~~ | ↷k | | | | |

```
k.acquire()    k.acquire()    k.acquire()
  0 = p.f        1 = p.f
  p.f = 0+1      p.f = 1+1
  p.g = 1        p.g = 2        y = p.g
k.release()    k.release()    k.release()
return 0       return 1       return y
```

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ?p.g=1 | ↷k | | | | |

```
k.acquire()    k.acquire()    k.acquire()
  0 = p.f        1 = p.f
  p.f = 0+1      p.f = 1+1
  p.g = 1        p.g = 2        y = p.g
k.release()    k.release()    k.release()
return 0       return 1       return y
```

# Controlling speculation: timeliness

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ~~?p.g=1~~ | ↷k | !p.f=2 | | | |

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       1 = p.f
  p.f = 0+1     p.f = 1+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return 1      return y
```

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ?p.g=1 | ↷k | !p.f=2 | | | |

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       1 = p.f
  p.f = 0+1     p.f = 1+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return 1      return y
```

# Controlling speculation: timeliness

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ~~?p.g=1~~ | ↷k | !p.f=2 | !p.g=2 | | |

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       1 = p.f
  p.f = 0+1     p.f = 1+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return 1      return y
```

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ?p.g=1 | ↷k | !p.f=2 | !p.g=2 | | |

```
k.acquire()   k.acquire()   k.acquire()
  0 = p.f       1 = p.f
  p.f = 0+1     p.f = 1+1
  p.g = 1       p.g = 2       y = p.g
k.release()   k.release()   k.release()
return 0      return 1      return y
```

# Controlling speculation: timeliness

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ~~?p.g=1~~ | ↷k | !p.f=2 | !p.g=2 | k↶ | |

```
k.acquire()     k.acquire()     k.acquire()
  0 = p.f         1 = p.f
  p.f = 0+1       p.f = 1+1
  p.g = 1         p.g = 2         y = p.g
k.release()     k.release()     k.release()
return 0        return 1        return y
```

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ↷k | !p.f=1 | !p.g=1 | k↶ |
| ?p.g=1 | ↷k | !p.f=2 | !p.g=2 | k↶ | |

```
k.acquire()     k.acquire()     k.acquire()
  0 = p.f         1 = p.f
  p.f = 0+1       p.f = 1+1
  p.g = 1         p.g = 2         y = p.g
k.release()     k.release()     k.release()
return 0        return 1        return y
```

# Controlling speculation: timeliness

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ↷k | ~~!p.f=1~~ | ~~!p.g=1~~ | k↶ |
| ~~?p.g=1~~ | ↷k | !p.f=2 | !p.g=2 | k↶ | ↷k |

```
k.acquire()    k.acquire()    k.acquire()
  0 = p.f        1 = p.f
  p.f = 0+1      p.f = 1+1
  p.g = 1        p.g = 2        y = p.g
k.release()    k.release()    k.release()
return 0       return 1       return y
```

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ↷k | ~~!p.f=1~~ | ~~!p.g=1~~ | k↶ |
| ?p.g=1 | ↷k | !p.f=2 | !p.g=2 | k↶ | ↷k |

```
k.acquire()    k.acquire()    k.acquire()
  0 = p.f        1 = p.f
  p.f = 0+1      p.f = 1+1
  p.g = 1        p.g = 2        y = p.g
k.release()    k.release()    k.release()
return 0       return 1       return y
```

# Controlling speculation: timeliness

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ↷k | ~~!p.f=1~~ | ~~!p.g=1~~ | k↶ |
| ~~?p.g=1~~ | ↷k | !p.f=2 | !p.g=2 | k↶ | ↷k |

```
k.acquire()     k.acquire()     k.acquire()
  0 = p.f         1 = p.f
  p.f = 0+1       p.f = 1+1
  p.g = 1         p.g = 2         2 = p.g
k.release()     k.release()     k.release()
return 0        return 1        return 2
```

| | | | | | |
|---|---|---|---|---|---|
| ~~!p.f=0~~ | ~~!p.g=0~~ | ↷k | ~~!p.f=1~~ | ~~!p.g=1~~ | k↶ |
| ?p.g=1 | ↷k | !p.f=2 | !p.g=2 | k↶ | ↷k |

```
k.acquire()     k.acquire()     k.acquire()
  0 = p.f         1 = p.f
  p.f = 0+1       p.f = 1+1
  p.g = 1         p.g = 2         1 = p.g
k.release()     k.release()     k.release()
return 0        return 1        return 1
```

# Controlling speculation: timeliness

| | | | | | | |
|---|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ↷k | !p.f=1 | !p.g=1 | k↶ | |
| ~~?p.g=1~~ | ↷k | !p.f=2 | !p.g=2 | k↶ | ↷k | k↶ |

```
k.acquire()    k.acquire()    k.acquire()
  0 = p.f        1 = p.f
  p.f = 0+1      p.f = 1+1
  p.g = 1        p.g = 2        2 = p.g
k.release()    k.release()    k.release()
return 0       return 1       return 2
```

| | | | | | | |
|---|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | ↷k | !p.f=1 | !p.g=1 | k↶ | |
| ?p.g=1 | ↷k | !p.f=2 | !p.g=2 | k↶ | ↷k | k↶ |

```
k.acquire()    k.acquire()    k.acquire()
  0 = p.f        1 = p.f
  p.f = 0+1      p.f = 1+1
  p.g = 1        p.g = 2        1 = p.g
k.release()    k.release()    k.release()
return 0       return 1       return 1
```

# Roach Motel

```
k.acquire()   k.acquire()   x = p.f              y = p.g
  p.f = 2       p.f = 1     k.acquire()          p.h = y
k.release()   k.release()     z = p.h            return y
                              if(x==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(x,z)
```

```
k.acquire()   k.acquire()   x = p.f              y = p.g
  p.f = 2       p.f = 1     k.acquire()          p.h = y
k.release()   k.release()     z = p.h            return y
                              if(x==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(x,z)
```

# Roach Motel

```
!p.f=0   !p.g=0   !p.h=0

k.acquire()   k.acquire()   x = p.f              y = p.g
  p.f = 2       p.f = 1     k.acquire()          p.h = y
k.release()   k.release()     z = p.h            return y
                              if(x==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(x,z)
```

```
!p.f=0   !p.g=0   !p.h=0

k.acquire()   k.acquire()   x = p.f              y = p.g
  p.f = 2       p.f = 1     k.acquire()          p.h = y
k.release()   k.release()     z = p.h            return y
                              if(x==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(x,z)
```

# Roach Motel

```
!p.f=0   !p.g=0   !p.h=0   ↷k
```

```
k.acquire()   k.acquire()   x = p.f                     y = p.g
  p.f = 2       p.f = 1     k.acquire()                 p.h = y
k.release()   k.release()     z = p.h                   return y
                              if(x==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(x,z)
```

```
!p.f=0   !p.g=0   !p.h=0   ↷k
```

```
k.acquire()   k.acquire()   x = p.f                     y = p.g
  p.f = 2       p.f = 1     k.acquire()                 p.h = y
k.release()   k.release()     z = p.h                   return y
                              if(x==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(x,z)
```

# Roach Motel

| | | | | |
|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 |

```
k.acquire()   k.acquire()   x = p.f                  y = p.g
  p.f = 2       p.f = 1     k.acquire()              p.h = y
k.release()   k.release()     z = p.h                return y
                              if(x==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(x,z)
```

| | | | | |
|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 |

```
k.acquire()   k.acquire()   x = p.f                  y = p.g
  p.f = 2       p.f = 1     k.acquire()              p.h = y
k.release()   k.release()     z = p.h                return y
                              if(x==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(x,z)
```

# Roach Motel

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ |

```
k.acquire()   k.acquire()   x = p.f                y = p.g
  p.f = 2       p.f = 1     k.acquire()            p.h = y
k.release()   k.release()     z = p.h              return y
                              if(x==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(x,z)
```

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ |

```
k.acquire()   k.acquire()   x = p.f                y = p.g
  p.f = 2       p.f = 1     k.acquire()            p.h = y
k.release()   k.release()     z = p.h              return y
                              if(x==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(x,z)
```

# Roach Motel

| | | | | | |
|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ |

```
k.acquire()   k.acquire()   2 = p.f                y = p.g
  p.f = 2       p.f = 1     k.acquire()            p.h = y
k.release()   k.release()     z = p.h              return y
                              if(2==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(2,z)
```

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ | ↷k |

```
k.acquire()   k.acquire()   x = p.f                y = p.g
  p.f = 2       p.f = 1     k.acquire()            p.h = y
k.release()   k.release()     z = p.h              return y
                              if(x==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(x,z)
```

# Roach Motel

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ | ↷k |

```
k.acquire()   k.acquire()   2 = p.f              y = p.g
  p.f = 2       p.f = 1     k.acquire()          p.h = y
k.release()   k.release()     z = p.h            return y
                              if(2==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(2,z)
```

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ | ↷k |
| !p.f=1 | | | | | | |

```
k.acquire()   k.acquire()   x = p.f              y = p.g
  p.f = 2       p.f = 1     k.acquire()          p.h = y
k.release()   k.release()     z = p.h            return y
                              if(x==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(x,z)
```

# Roach Motel

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ | ↷k |
| !p.f=1 | | | | | | |

```
k.acquire()    k.acquire()    2 = p.f                   y = p.g
  p.f = 2        p.f = 1      k.acquire()               p.h = y
k.release()    k.release()      z = p.h                 return y
                                if(2==2) p.g = 1
                                else     p.g = z
                              k.release()
                              return(2,z)
```

| | | | | | | |
|---|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ | ↷k |
| !p.f=1 | k↶ | | | | | |

```
k.acquire()    k.acquire()    x = p.f                   y = p.g
  p.f = 2        p.f = 1      k.acquire()               p.h = y
k.release()    k.release()      z = p.h                 return y
                                if(x==2) p.g = 1
                                else     p.g = z
                              k.release()
                              return(x,z)
```

# Roach Motel

| | | | | | | |
|---|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ | ↷k |
| !p.f=1 | k↶ | | | | | |

```
k.acquire()   k.acquire()   2 = p.f                y = p.g
  p.f = 2       p.f = 1     k.acquire()            p.h = y
k.release()   k.release()     z = p.h              return y
                              if(2==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(2,z)
```

| | | | | | | |
|---|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ | ↷k |
| !p.f=1 | k↶ | | | | | |

```
k.acquire()   k.acquire()   1 = p.f                y = p.g
  p.f = 2       p.f = 1     k.acquire()            p.h = y
k.release()   k.release()     z = p.h              return y
                              if(1==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(1,z)
```

# Roach Motel

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | ~~!p.f=2~~ | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | | | | |

```
    k.acquire()    k.acquire()    2 = p.f              y = p.g
      p.f = 2        p.f = 1      k.acquire()          p.h = y
    k.release()    k.release()      z = p.h            return y
                                    if(2==2) p.g = 1
                                    else     p.g = z
                                  k.release()
                                  return(2,z)
```

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | ~~!p.f=2~~ | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | | | | |

```
    k.acquire()    k.acquire()    1 = p.f              y = p.g
      p.f = 2        p.f = 1      k.acquire()          p.h = y
    k.release()    k.release()      z = p.h            return y
                                    if(1==2) p.g = 1
                                    else     p.g = z
                                  k.release()
                                  return(1,z)
```

# Roach Motel

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | ~~!p.f=2~~ | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ~~?p.h=1~~ | | | |

```
k.acquire()   k.acquire()   2 = p.f                      y = p.g
  p.f = 2       p.f = 1     k.acquire()                  p.h = y
k.release()   k.release()     z = p.h                    return y
                              if(2==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(2,z)
```

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | ~~!p.f=2~~ | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ?p.h=1 | | | |

```
k.acquire()   k.acquire()   1 = p.f                      y = p.g
  p.f = 2       p.f = 1     k.acquire()                  p.h = y
k.release()   k.release()     z = p.h                    return y
                              if(1==2) p.g = 1
                              else     p.g = z
                            k.release()
                            return(1,z)
```

# Roach Motel

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | ~~!p.f=2~~ | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ~~?p.h=1~~ | | | |

```
k.acquire()
  p.f = 2
k.release()
```

```
k.acquire()
  p.f = 1
k.release()
```

```
2 = p.f
k.acquire()
  z = p.h
  if(2==2) p.g = 1
  else     p.g = z
k.release()
return(2,z)
```

```
y = p.g
p.h = y
return y
```

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | ~~!p.f=2~~ | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ?p.h=1 | | | |

```
k.acquire()
  p.f = 2
k.release()
```

```
k.acquire()
  p.f = 1
k.release()
```

```
1 = p.f
k.acquire()
  z = p.h
  if(1==2) p.g = 1
  else     p.g = z
k.release()
return(1,z)
```

```
y = p.g
p.h = y
return y
```

# Roach Motel

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | ~~!p.f=2~~ | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ~~?p.h=1~~ | | | |

```
k.acquire()   k.acquire()   2 = p.f              y = p.g
  p.f = 2       p.f = 1     k.acquire()          p.h = y
k.release()   k.release()     0 = p.h            return y
                              if(2==2) p.g = 1
                              else     p.g = 0
                            k.release()
                            return(2,0)
```

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | ~~!p.f=2~~ | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ?p.h=1 | | | |

```
k.acquire()   k.acquire()   1 = p.f              y = p.g
  p.f = 2       p.f = 1     k.acquire()          p.h = y
k.release()   k.release()     1 = p.h            return y
                              if(1==2) p.g = 1
                              else     p.g = 1
                            k.release()
                            return(1,1)
```

# Roach Motel

| | | | | | | |
|---|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ~~?p.h=1~~ | !p.g=1 | | |

```
k.acquire()   k.acquire()   2 = p.f                  y = p.g
  p.f = 2       p.f = 1     k.acquire()              p.h = y
k.release()   k.release()     0 = p.h                return y
                              if(2==2) p.g = 1
                              else     p.g = 0
                            k.release()
                            return(2,0)
```

| | | | | | | |
|---|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ?p.h=1 | !p.g=1 | | |

```
k.acquire()   k.acquire()   1 = p.f                  y = p.g
  p.f = 2       p.f = 1     k.acquire()              p.h = y
k.release()   k.release()     1 = p.h                return y
                              if(1==2) p.g = 1
                              else     p.g = 1
                            k.release()
                            return(1,1)
```

# Roach Motel

| | | | | | | |
|---|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ~~?p.h=1~~ | !p.g=1 | | |

```
    k.acquire()   k.acquire()   2 = p.f               1 = p.g
      p.f = 2       p.f = 1     k.acquire()           p.h = 1
    k.release()   k.release()     0 = p.h             return 1
                                  if(2==2) p.g = 1
                                  else     p.g = 0
                                k.release()
                                return(2,0)
```

| | | | | | | |
|---|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ?p.h=1 | !p.g=1 | | |

```
    k.acquire()   k.acquire()   1 = p.f               1 = p.g
      p.f = 2       p.f = 1     k.acquire()           p.h = 1
    k.release()   k.release()     1 = p.h             return 1
                                  if(1==2) p.g = 1
                                  else     p.g = 1
                                k.release()
                                return(1,1)
```

# Roach Motel

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | ~~!p.f=2~~ | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ~~?p.h=1~~ | !p.g=1 | !p.h=1 | |

```
   k.acquire()  k.acquire()  2 = p.f                  1 = p.g
     p.f = 2      p.f = 1    k.acquire()              p.h = 1
   k.release()  k.release()    0 = p.h                return 1
                               if(2==2) p.g = 1
                               else     p.g = 0
                             k.release()
                             return(2,0)
```

| | | | | | | |
|---|---|---|---|---|---|---|
| ~~!p.f=0~~ | !p.g=0 | !p.h=0 | ↷k | ~~!p.f=2~~ | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ?p.h=1 | !p.g=1 | !p.h=1 | |

```
   k.acquire()  k.acquire()  1 = p.f                  1 = p.g
     p.f = 2      p.f = 1    k.acquire()              p.h = 1
   k.release()  k.release()    1 = p.h                return 1
                               if(1==2) p.g = 1
                               else     p.g = 1
                             k.release()
                             return(1,1)
```

# Roach Motel

| | | | | | | |
|---|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ~~?p.h=1~~ | !p.g=1 | !p.h=1 | k↶ |

```
k.acquire()   k.acquire()   2 = p.f                    1 = p.g
  p.f = 2       p.f = 1     k.acquire()                p.h = 1
k.release()   k.release()     0 = p.h                  return 1
                              if(2==2) p.g = 1
                              else     p.g = 0
                            k.release()
                            return(2,0)
```

| | | | | | | |
|---|---|---|---|---|---|---|
| !p.f=0 | !p.g=0 | !p.h=0 | ↷k | !p.f=2 | k↶ | ↷k |
| !p.f=1 | k↶ | ↷k | ?p.h=1 | !p.g=1 | !p.h=1 | k↶ |

```
k.acquire()   k.acquire()   1 = p.f                    1 = p.g
  p.f = 2       p.f = 1     k.acquire()                p.h = 1
k.release()   k.release()     1 = p.h                  return 1
                              if(1==2) p.g = 1
                              else     p.g = 1
                            k.release()
                            return(1,1)
```