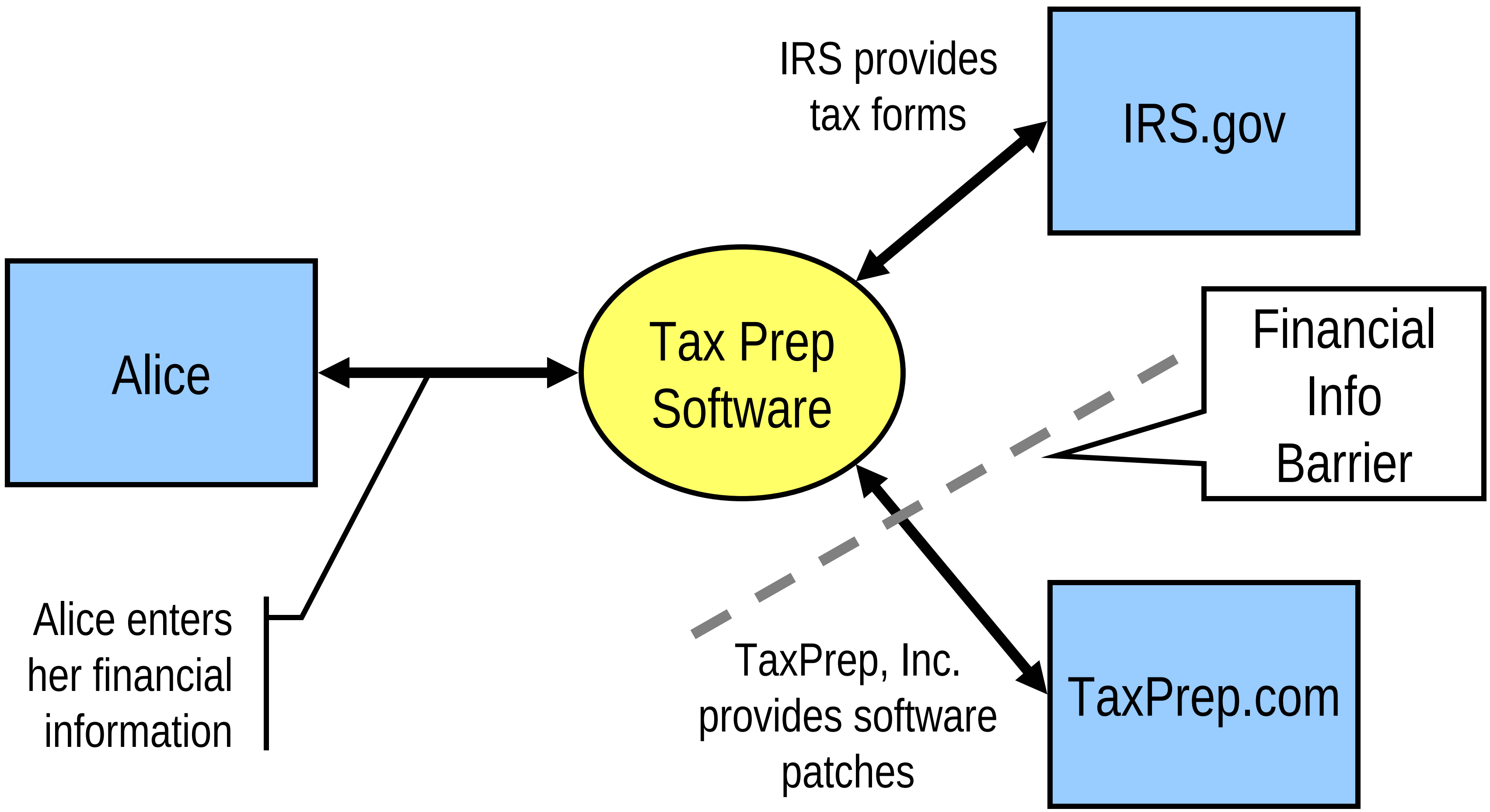
RIFLE: An Architectural Framework for User-Centric Information-Flow Security

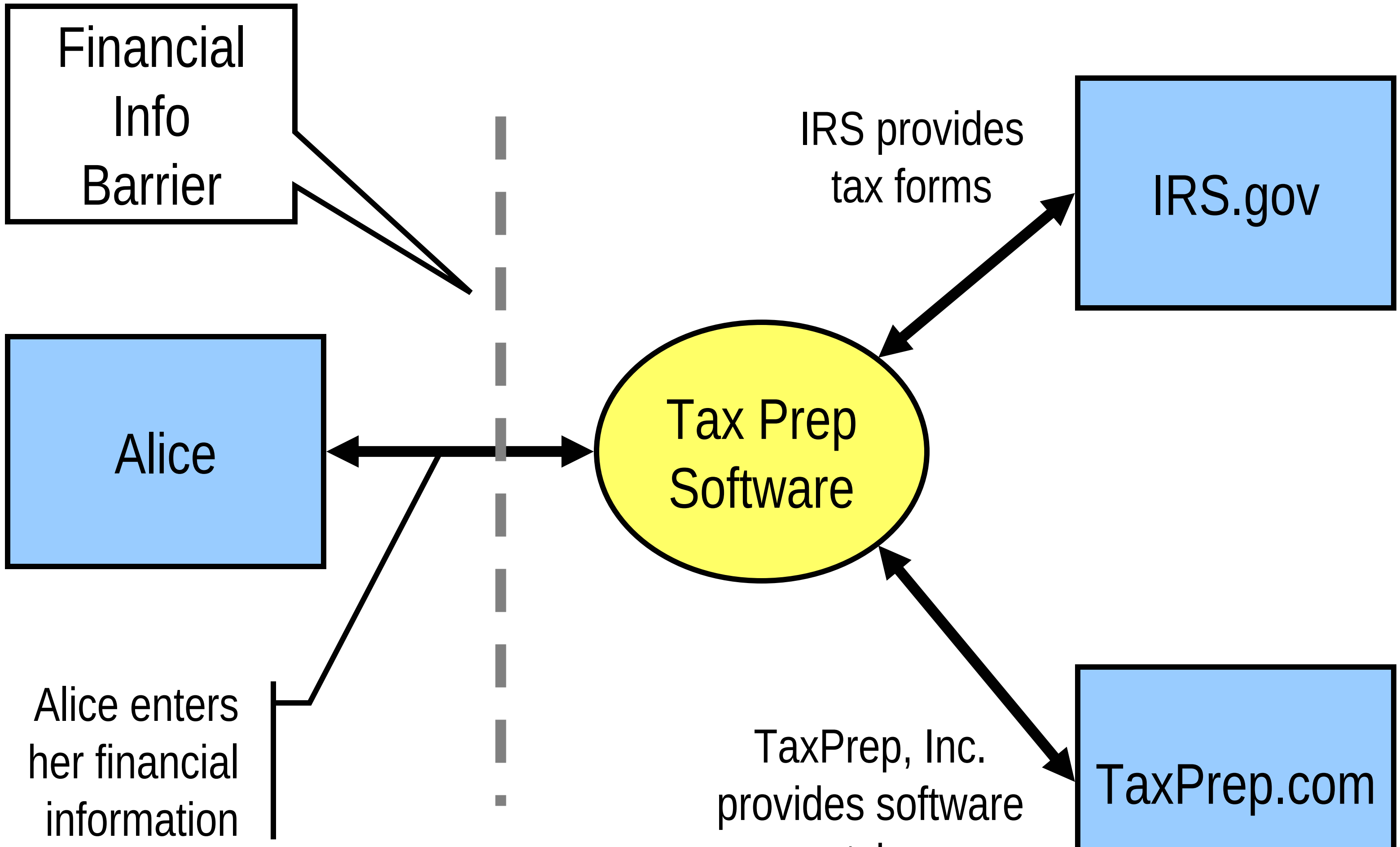
Neil Vachharajani · Matthew J. Bridges
Jonathan Chang · Ram Rangan
Guilherme Ottoni · Jason A. Blome · George A. Reis
Manish Vachharajani · David I. August

Liberty Research Group
Princeton University

Information-Flow Security in the Real World



Information-Flow Security in the Real World



Information-Flow Security in the Real World

All programs must be assumed unsafe

- **Malicious** programs intentionally leak information
- **Buggy** programs that unintentionally leak information

User-Centric Information-Flow Security

1. Users want to establish their *own* security policy
   - CIA's security needs differ from Joe Average's
2. Users want data-dependent security policies
   - Web browser with web search form data
   - Web browser with banking login form data
3. Users should not have to sacrifice security for functionality
   - All programs should be secure or securable
   - Only security holes that will be realized are significant

Definition of Security: Non-Interference

Integrity

- Untrusted inputs should not affect trusted outputs
- Example: prevent input from being executed [Suh 04, Crandall 04]

Confidentiality [Denning 76, Myers 97, Myers 99, Tse 04]

- High security inputs should not affect low security outputs
- Example: tax preparation software

Key mechanism: tracking flow of information through code

- Integrity/confidentiality are dual
- Policies and enforcement rely on information flow

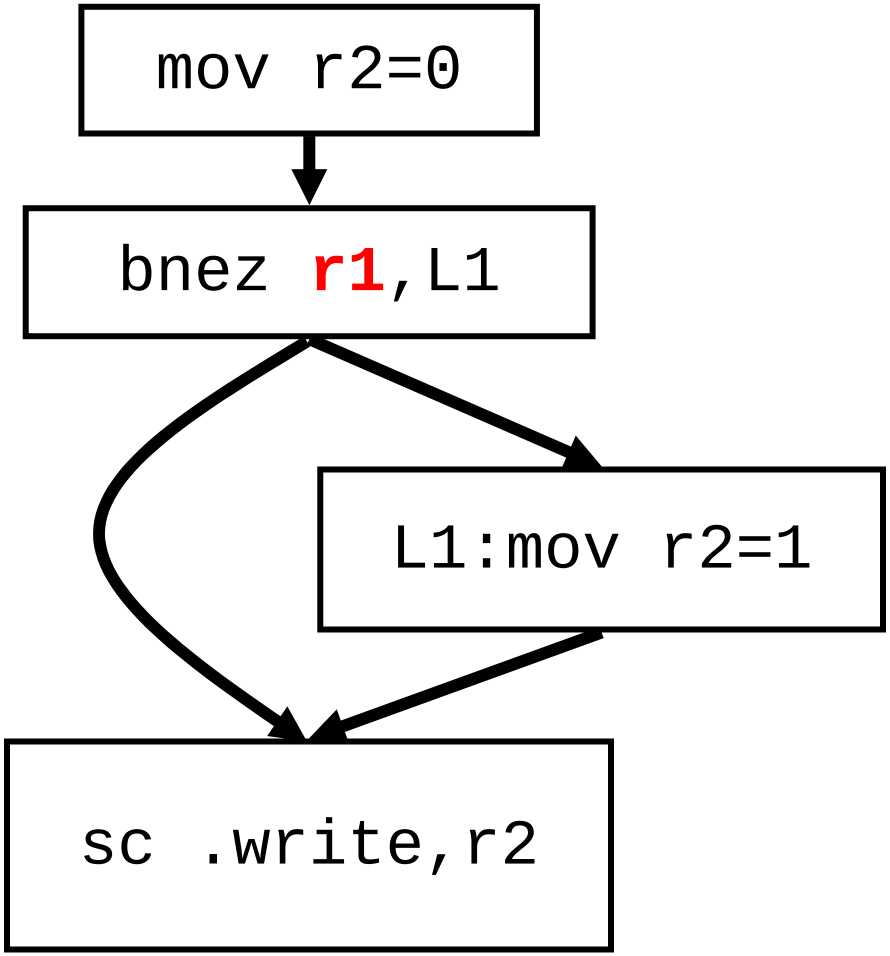
Information-Flow Security: Tainting Data

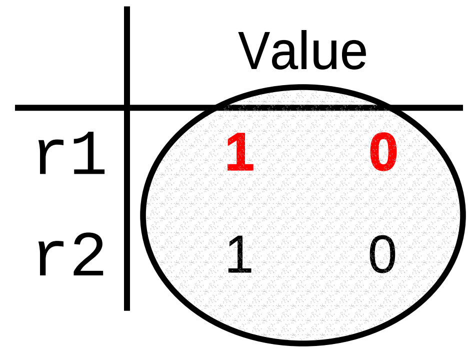
- Used in Perl's "taint" mode and other works
  [Denning 76, Suh 04, Crandall 04]

```
add r4 = r1,r2
add r5 = r4,r3
div r6 = r5, 3
sc  .write, r6
```

1. Program inputs are **tainted** or **labeled** with a security class
2. Labels propagate through computation
3. Certain operations enforce a security policy by verifying operand labels for security

Problems with the Taint Solution





Control Flow Can Leak Information!

User-Centric Information-Flow Security

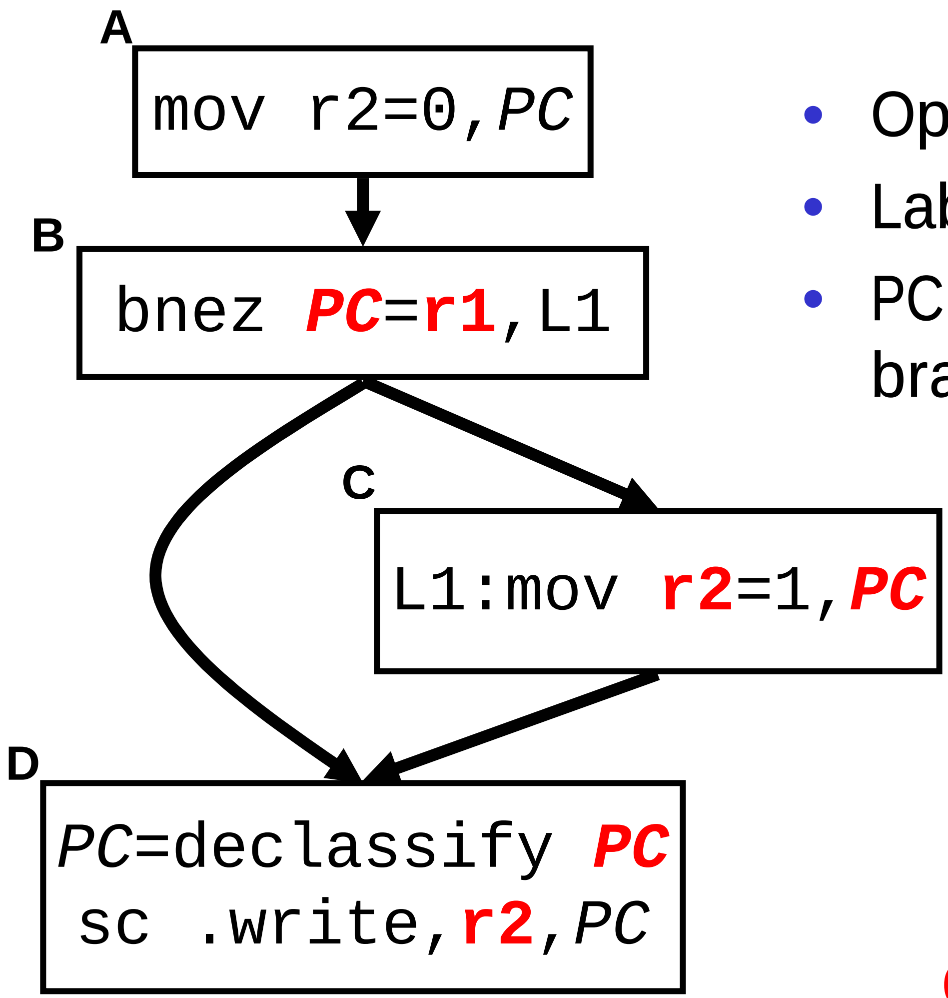
Essential for User-Centric IFS

Deal breaker for User-Centric IFS

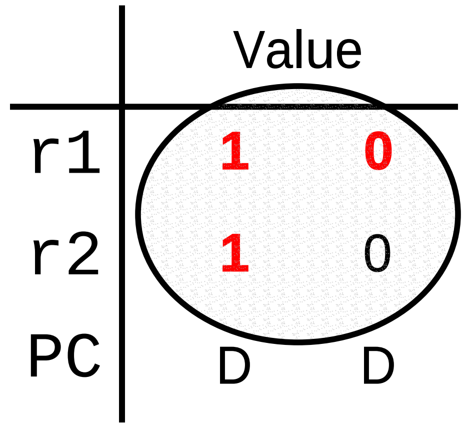
Fundamentally Impossible

	Leaks Information	Policy Enforcement	Programmer Support	Conservative	Performance Loss
Taint [Suh 2004]	~~Yes~~	Dynamic	No	No	Moderate
Static Systems [Denning 76, Myers 97, Myers 99]	Rate Limited	~~Static~~	~~Yes~~	Yes	None
Static with Runtime Principles [Tse 2004]	Rate Limited	Hybrid	~~Yes~~	Yes	Little
Ideal User-Centric	~~No~~	Dynamic/ Hybrid	No	~~No~~	None
RIFLE	Rate Limited	Dynamic	No	Yes	Moderate

Naïve "Solution": Taint the Program Counter

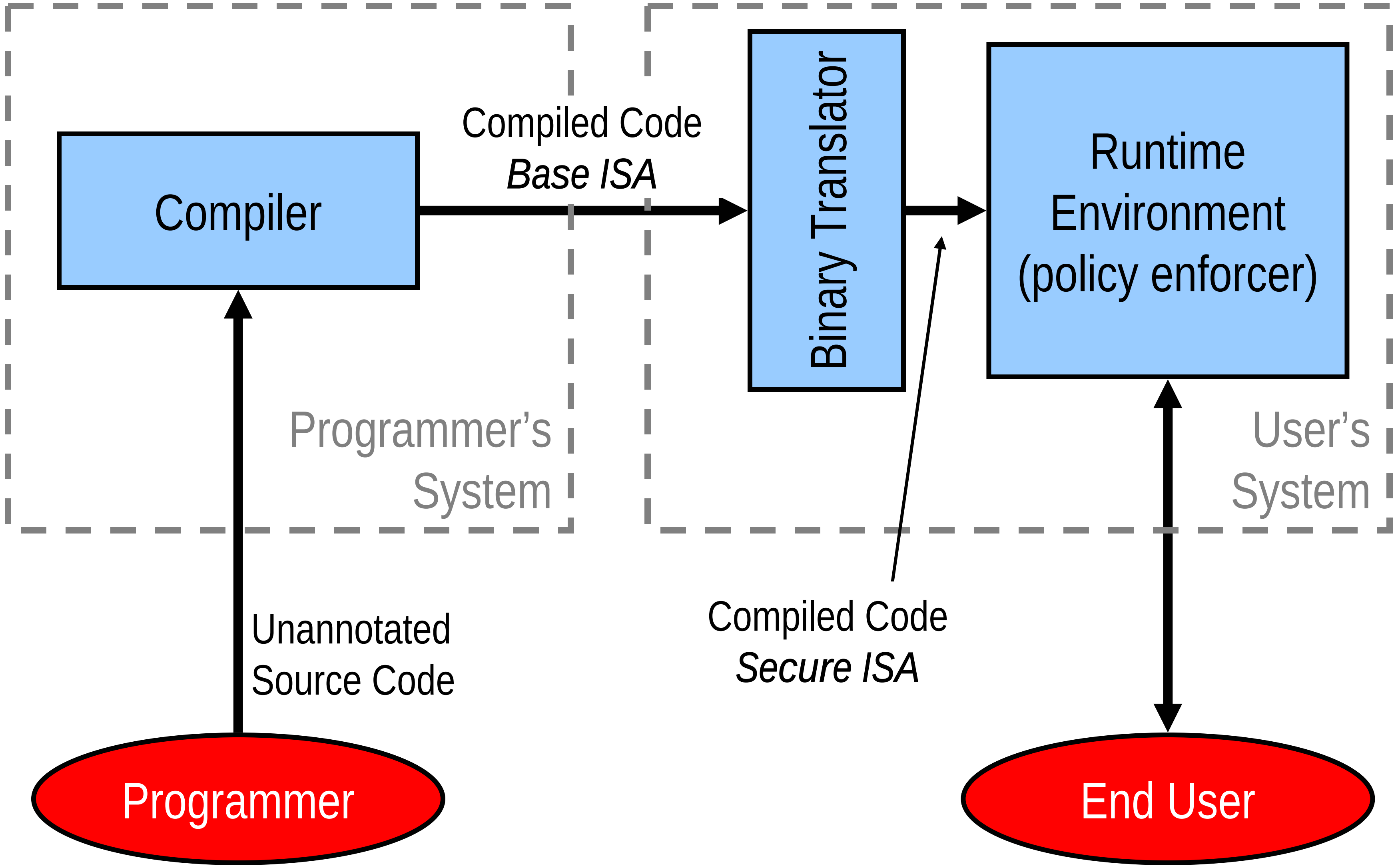


- Ops have *implicit* PC operand
- Label PC like other operands
- PC should be declassified after branch merge

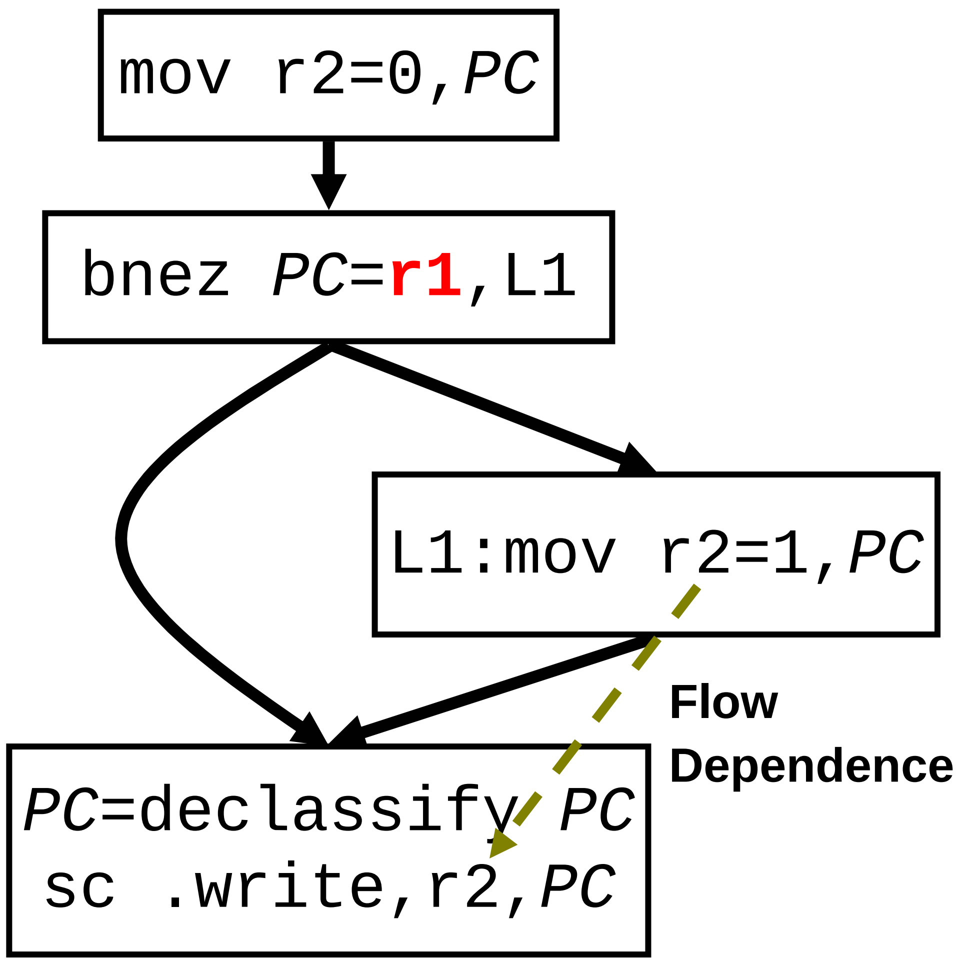


Code can leak information whether it is executed or not!

RIFLE: The Big Picture

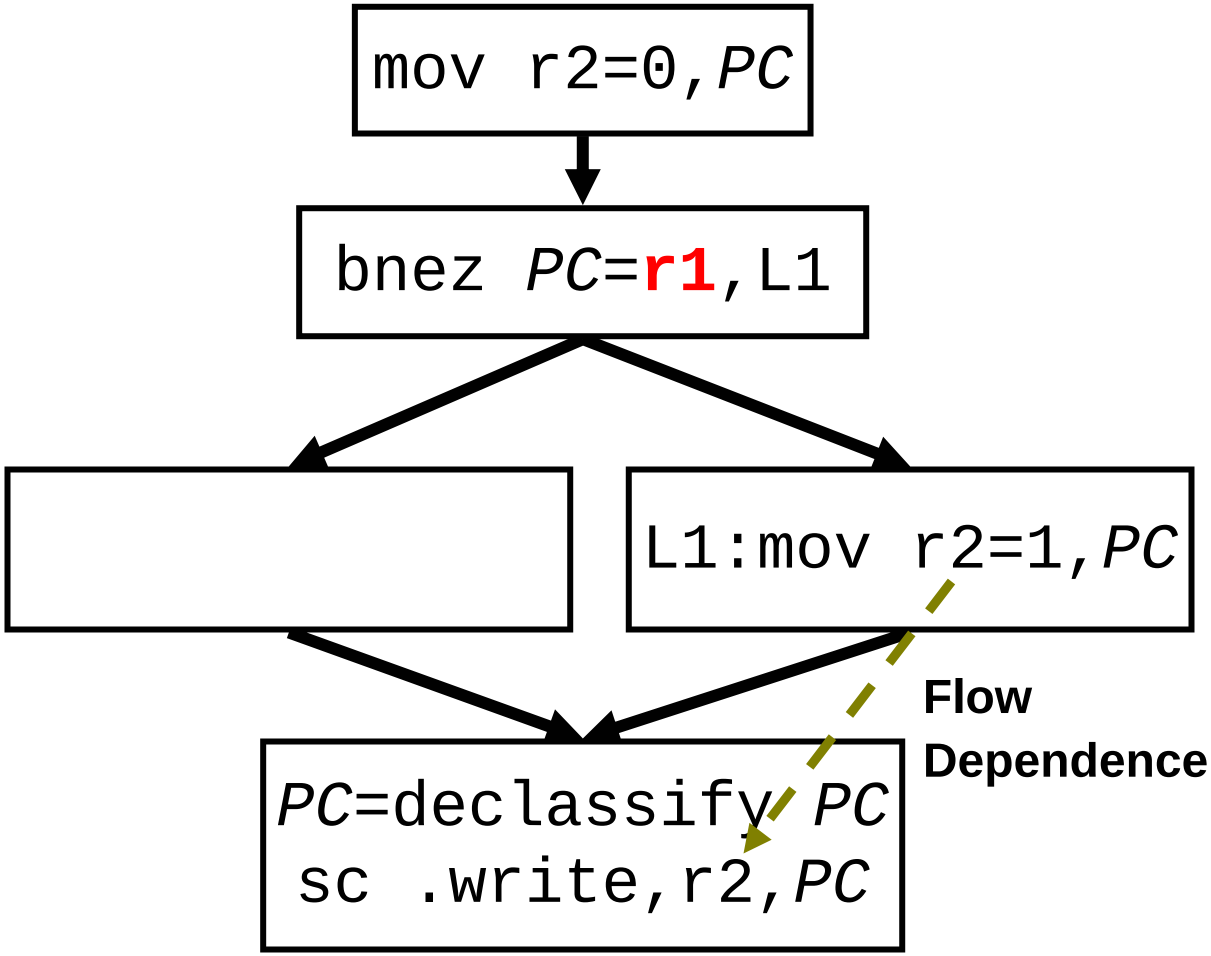


Naïve Binary Translation



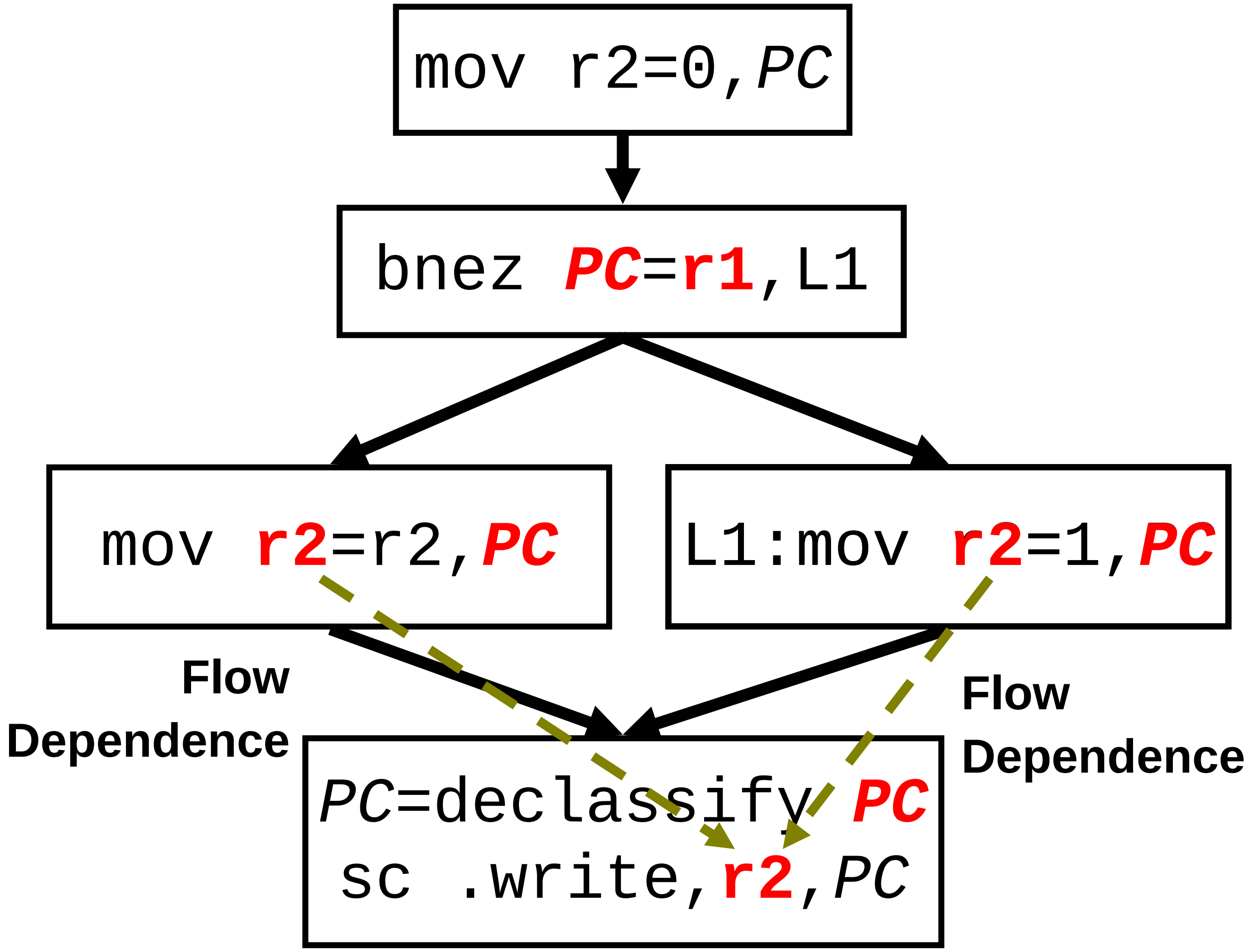
Naïve Binary Translation

1. Force every `if` to have an `else`



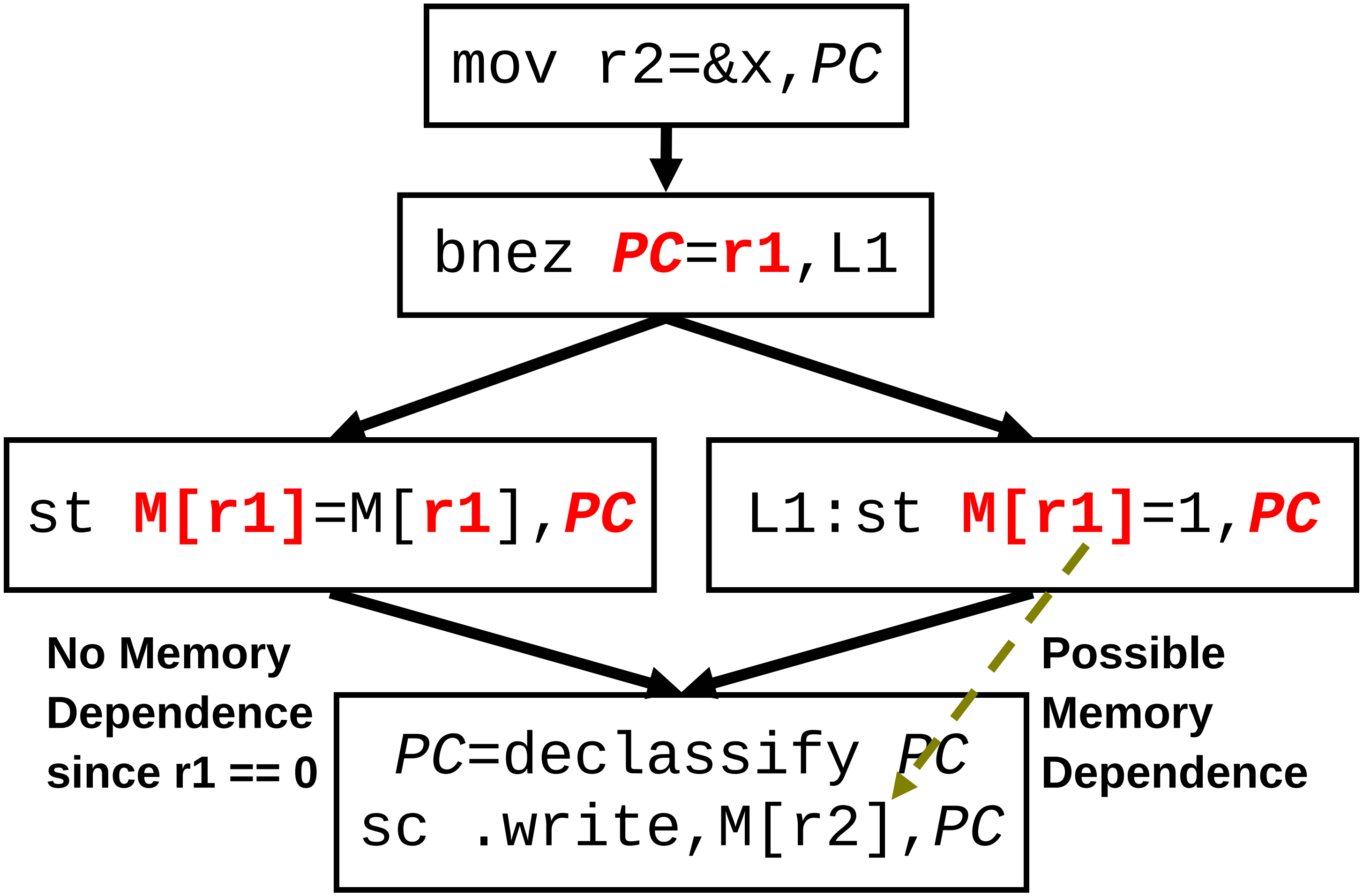
Naïve Binary Translation

1. Force every `if` to have an `else`
2. On each side of the branch, modify same variables



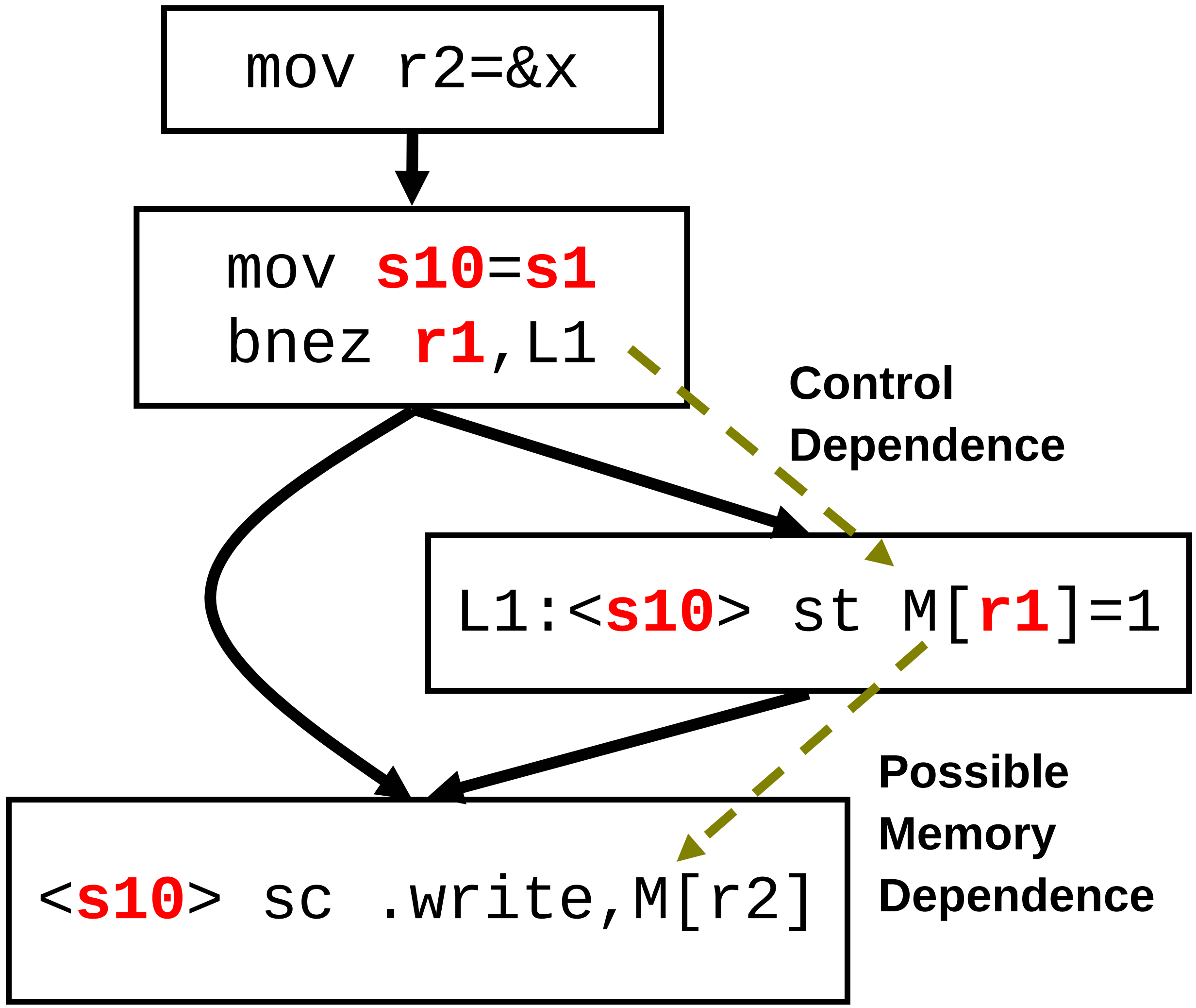
Naïve Binary Translation

- But, what about memory?



RIFLE Binary Translation

Key Insight: Handle implicit flows at data use, not data definition.



Results: Security

Word Count (wc)

- Function calls and returns
- Global pointer, stack pointer

```
stdout
   239     717    7834 wc.NM
  1183    2856   49918 wc.MAP
   593    3532   28025 load6.txt
  4445   25616  188549 src4+2or6.txt
  6460   32721  274326 total
```

- wc.NM
- wc.MAP
- load6.txt
- src4+2or6.txt
- Inputs Combined
- Command Line
- Program Binary

PGP – identified unexpected information flows!

- Key ring – each key labeled with a unique label
- Plain text – colored with a unique label
- Cipher text –
  - Expected: labeled with key's label and plain text label
  - Actual: labeled with label of all keys up to used key and plain text label

Hardware Implementation & Optimizations

- All instructions create explicit flows
  - Use shadow registers/memory to store security labels
  - Augment processor data path to track explicit flows
- Transformation inserts redundant security register defines
  - Many instructions added
  - Many security registers needed

```
    add r1=0,1
    mov s50 = s10
    mov s60 = s50
    (r10) jump L2
L1: <s50>(r1) jump L3
…
L2: <s60> add r1=0,0
    jump L1
```

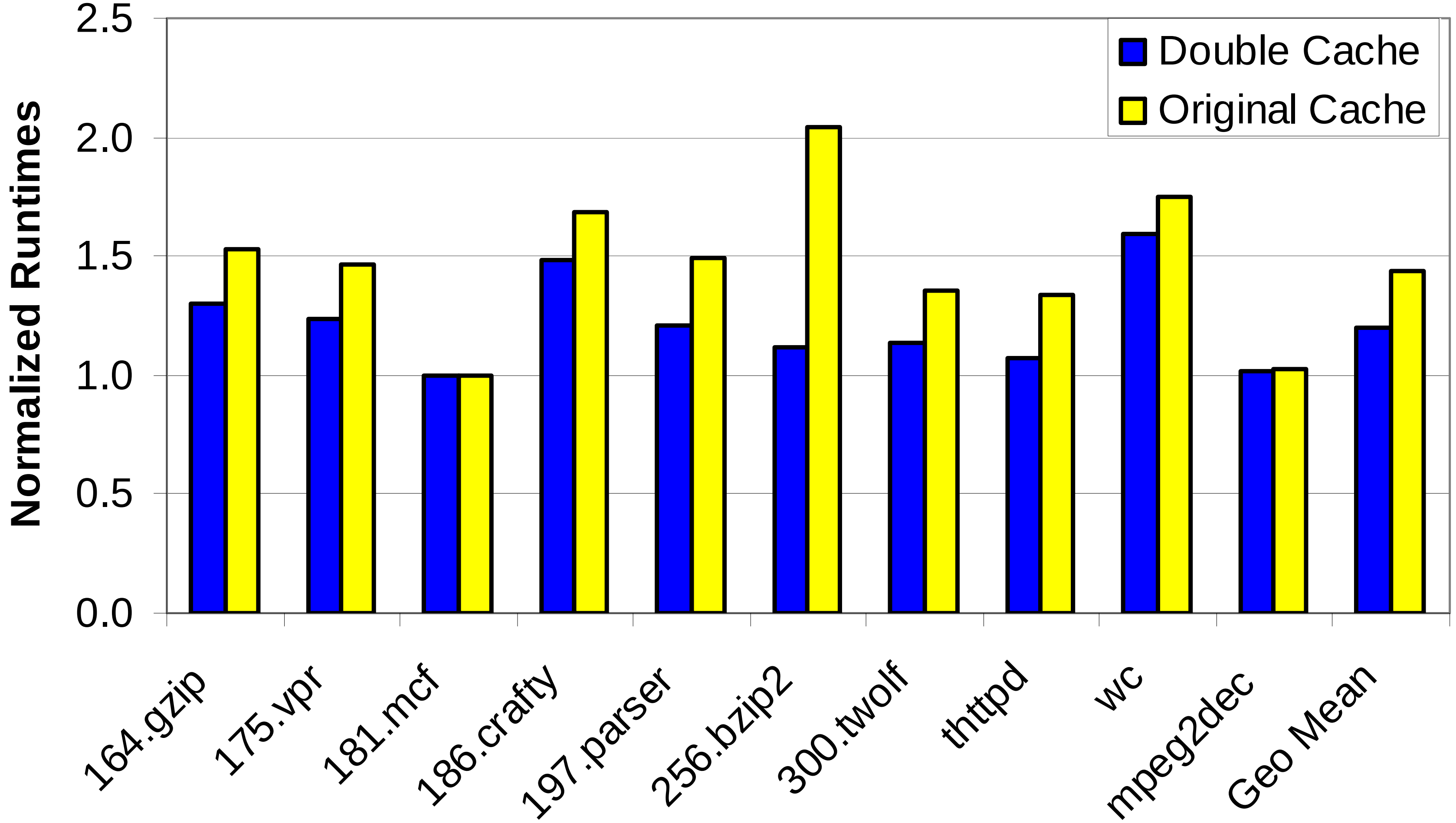
Before Opti

```
    add r1=0,1
    s50 = mov s10
    (r10) jump L2
L1: <s50>(r1) jump L3
…
L2: <s50> add r1=0,0
    jump L1
```

After Opti

Results: Performance

Validated Itanium 2 model built in the Liberty Simulation Environment



Conclusions & Future Work

- User-centric information flow security empowers *users*
  - User (not programmer) tailored security policy
  - Data-based (not program-based) security
  - Any program (no need for special languages) can be secured
- User-centric information flow security is possible
- RIFLE provides user-centric information-flow security by:
  - Tracking flow and enforcing policies *dynamically*
  - Using static “hints” via binary translation to establish security
- Future work
  - Improved performance – more optimization, hardware acceleration
  - JVM implementation – for broadened applicability
  - Declassification – allowing user-controlled data “leaks”