

Eliminating cache-based timing attacks with instruction-based scheduling

*Deian Stefan, Pablo Buiras, Edward Z. Yang, Amit Levy,
David Terei, Alejandro Russo, and David Mazières*



STANFORD
UNIVERSITY

CHALMERS

Motivation: IFC Web platforms



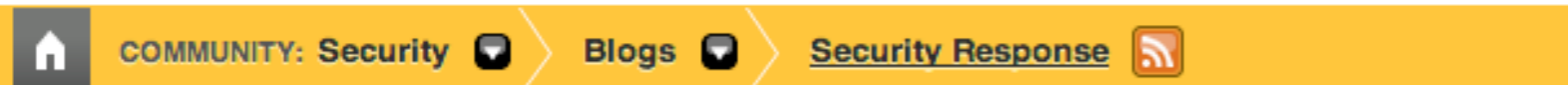
github:develop

Platforms allow 3rd-party developers to build apps that use our personal data

- Extend the websites beyond original intent!



Motivation: IFC Web platforms



Facebook Applications Accidentally Leaking Access to Third Parties - Updated

Platforms allow 3rd-party developers to build apps that use our personal data
^and sometimes leak

- Extend the websites beyond original intent!



Path Uploads Your Entire iPhone Contact List By Default

The GitHub Blog

March 4, 2012 mojombo

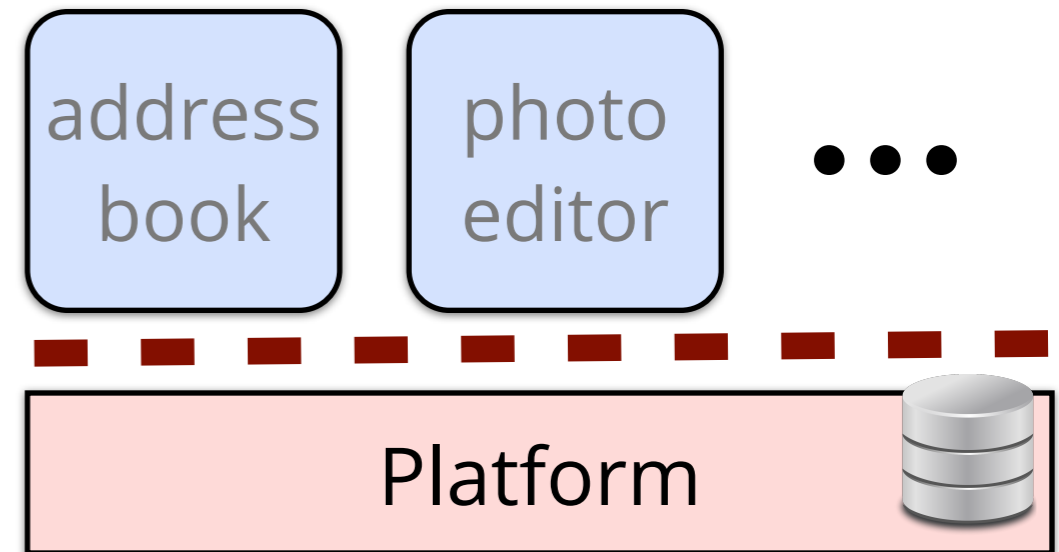
Public Key Security Vulnerability a

Motivation: IFC Web platforms

Challenge: can we ensure apps don't leak data?

Current approach: DAC

- Restrict what data app can access
- 👎 Cannot guarantee what app does with data



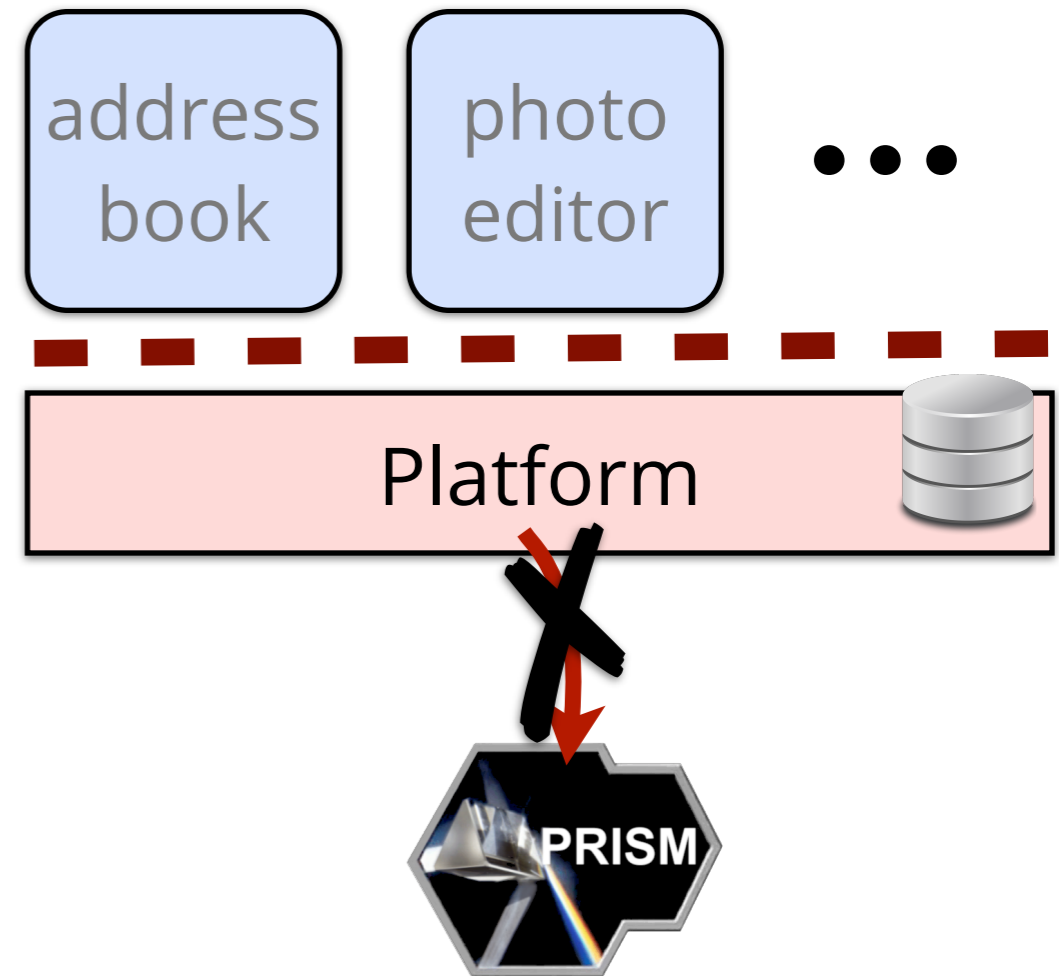
Motivation: IFC Web platforms

Challenge: can we ensure apps don't leak data?

Current approach: DAC

- Restrict what data app can access

- 👎 Cannot guarantee what app does with data

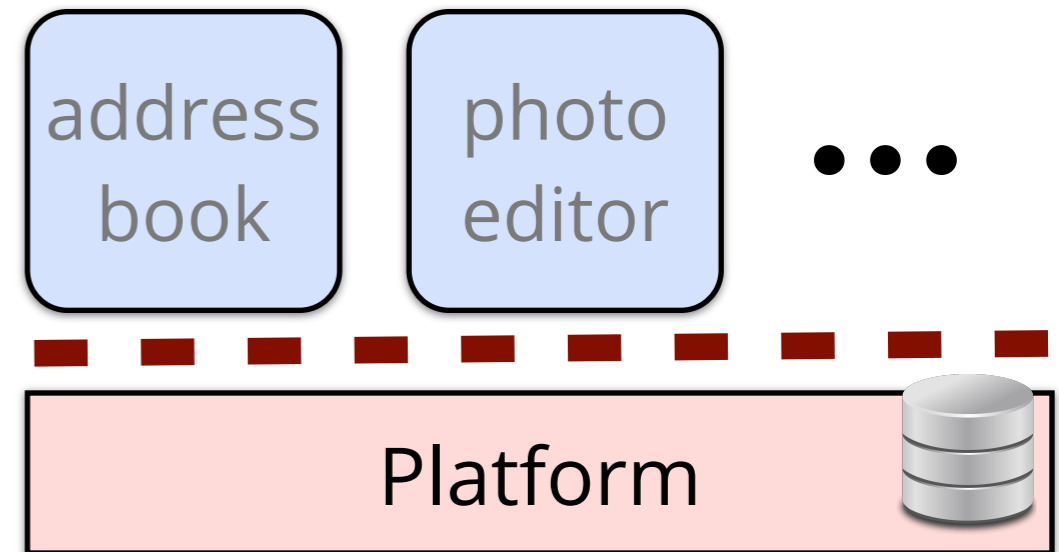


Motivation: IFC Web platforms

Challenge: can we ensure apps don't leak data?

Current approach: DAC

- Restrict what data app can access
- 👎 Cannot guarantee what app does with data

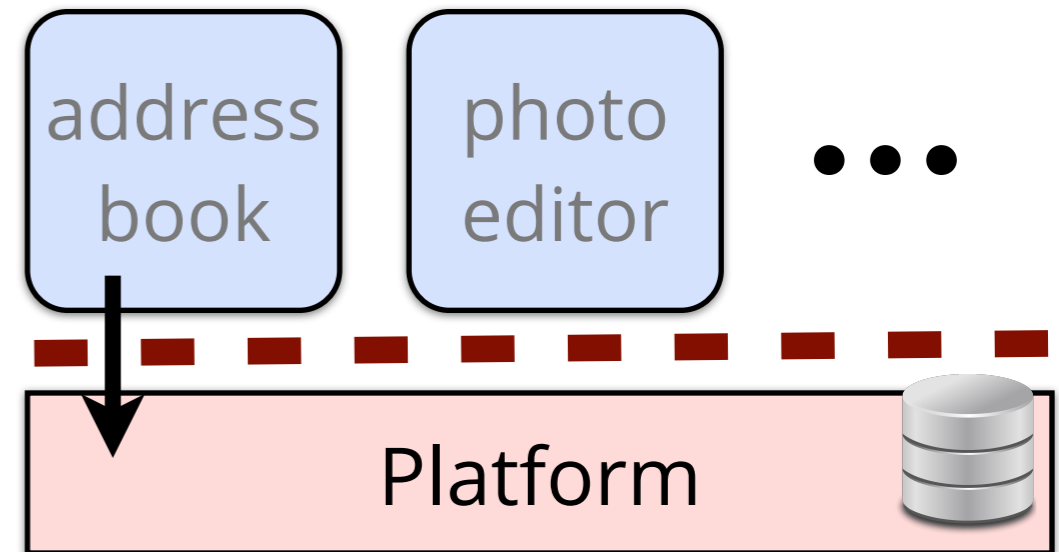


Motivation: IFC Web platforms

Challenge: can we ensure apps don't leak data?

Current approach: DAC

- Restrict what data app can access
- 👎 Cannot guarantee what app does with data



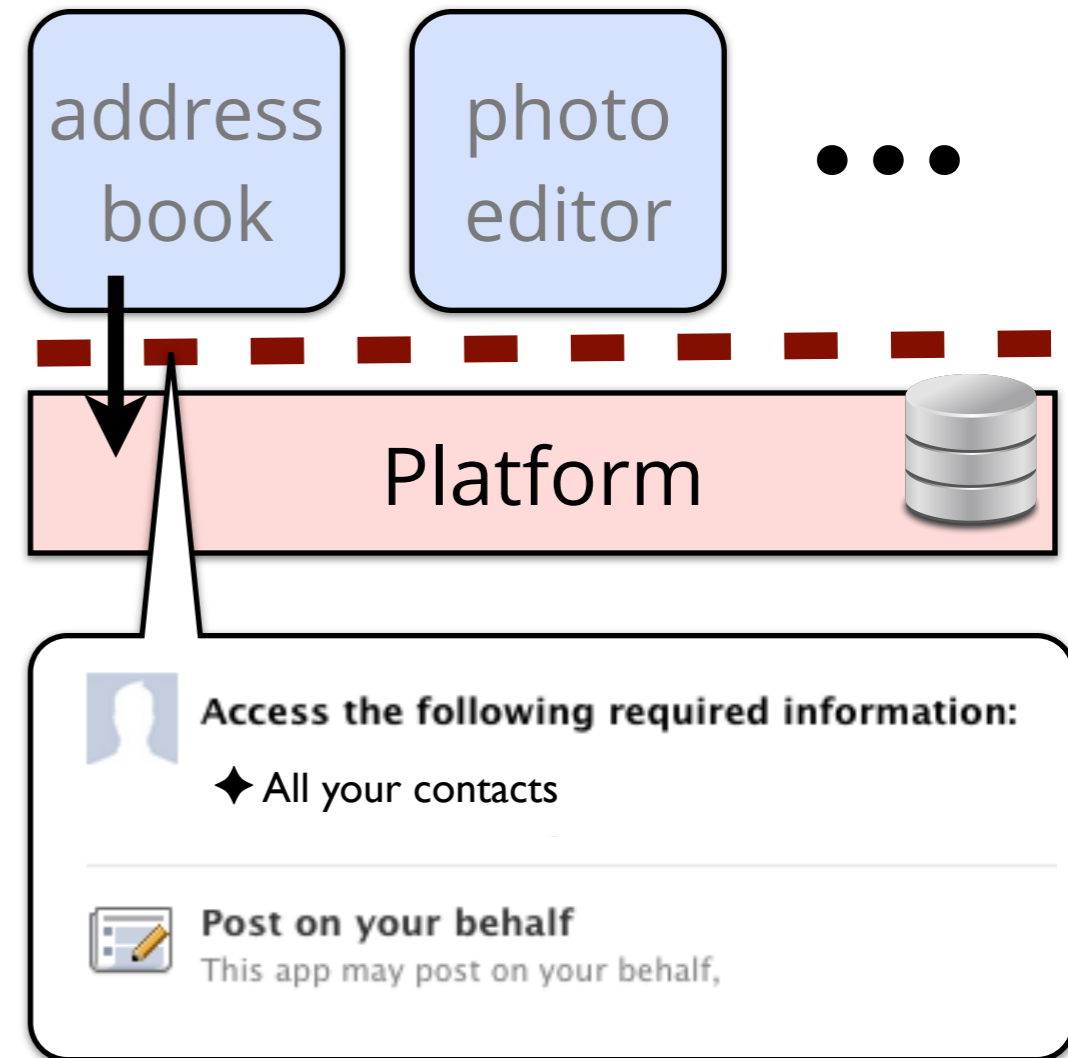
Motivation: IFC Web platforms

Challenge: can we ensure apps don't leak data?

Current approach: DAC

➤ Restrict what data app can access

👎 Cannot guarantee what app does with data



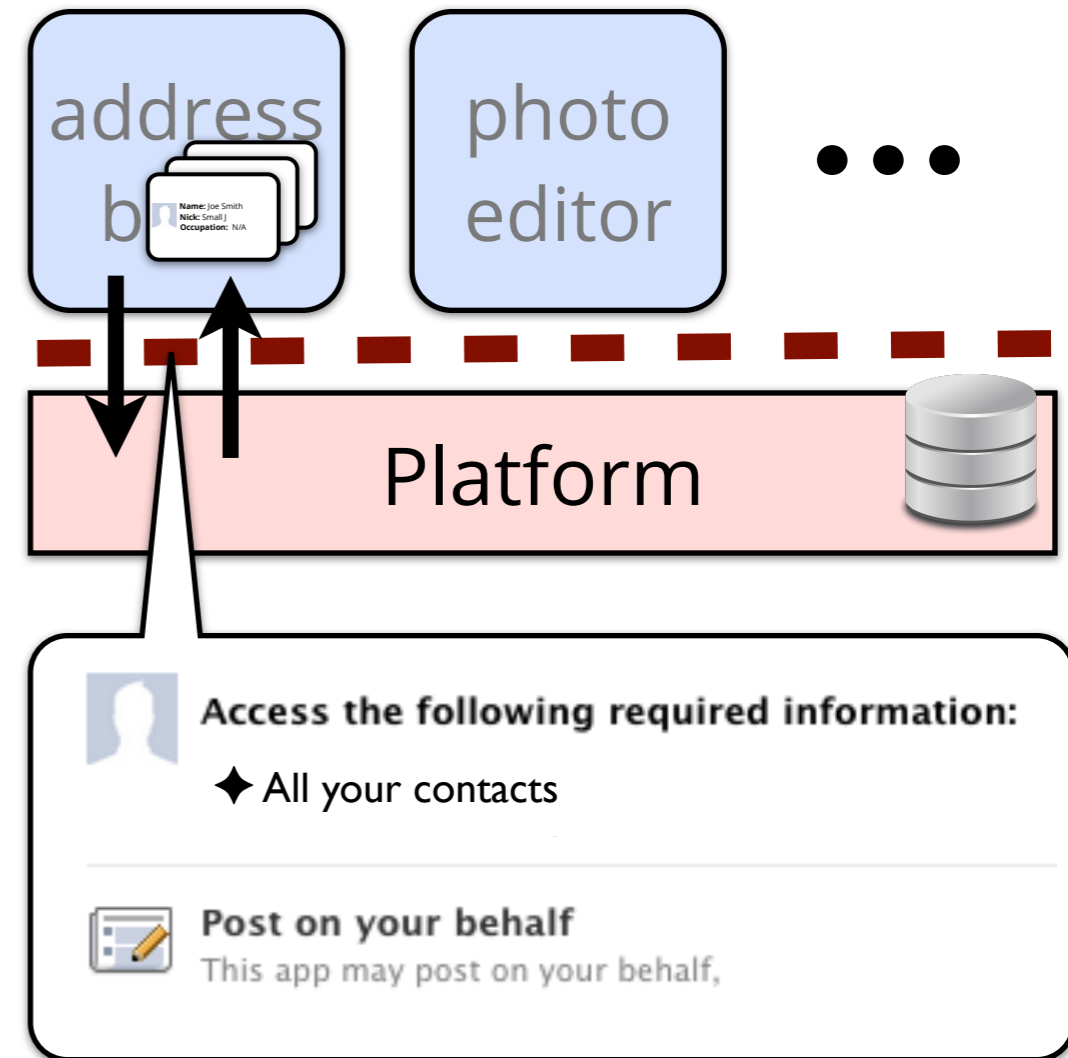
Motivation: IFC Web platforms

Challenge: can we ensure apps don't leak data?

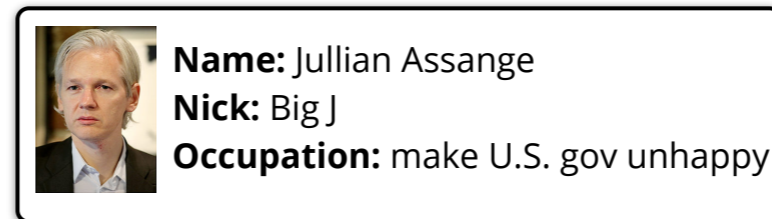
Current approach: DAC

- Restrict what data app can access

- 👎 Cannot guarantee what app does with data



Motivation: IFC Web platforms



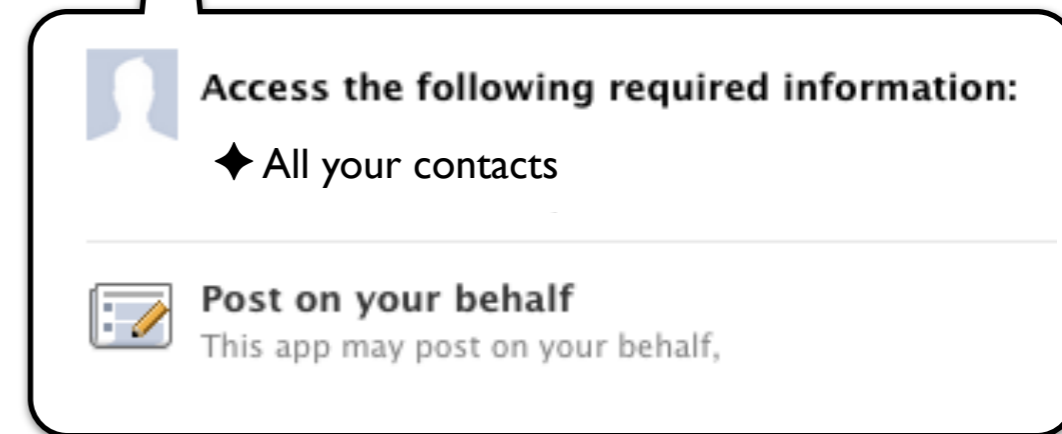
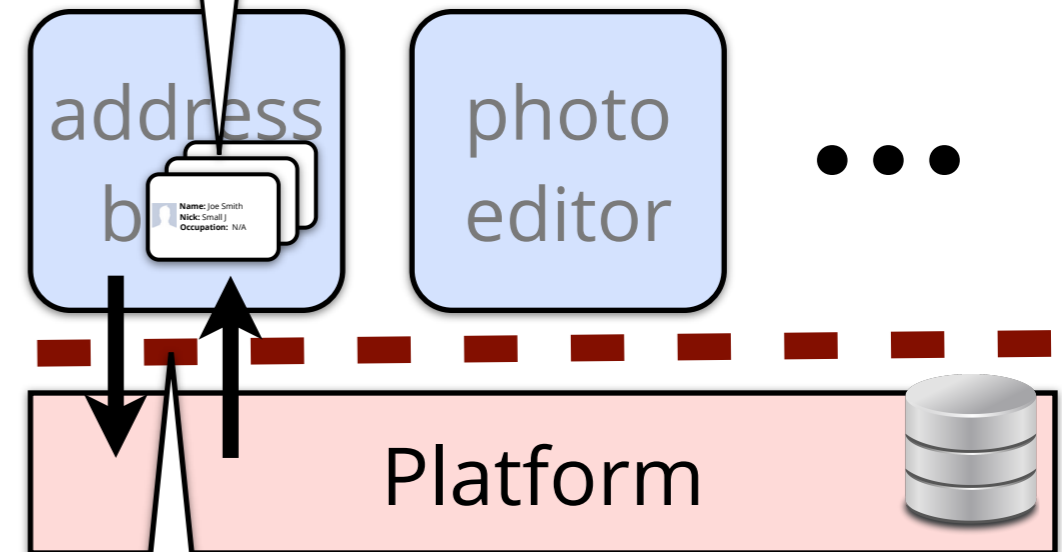
Name: Jullian Assange
Nick: Big J
Occupation: make U.S. gov unhappy

Challenge: can we ensure apps don't leak data?

Current approach: DAC

➤ Restrict what data app can access

👎 Cannot guarantee what app does with data

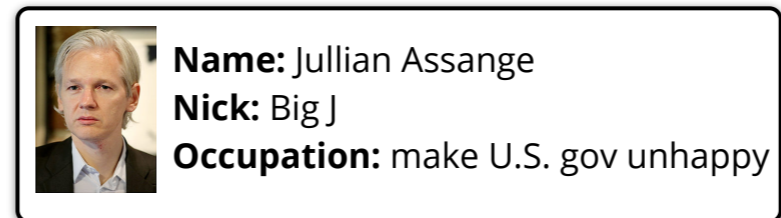


Access the following required information:

- ◆ All your contacts

Post on your behalf
This app may post on your behalf,

Motivation: IFC Web platforms



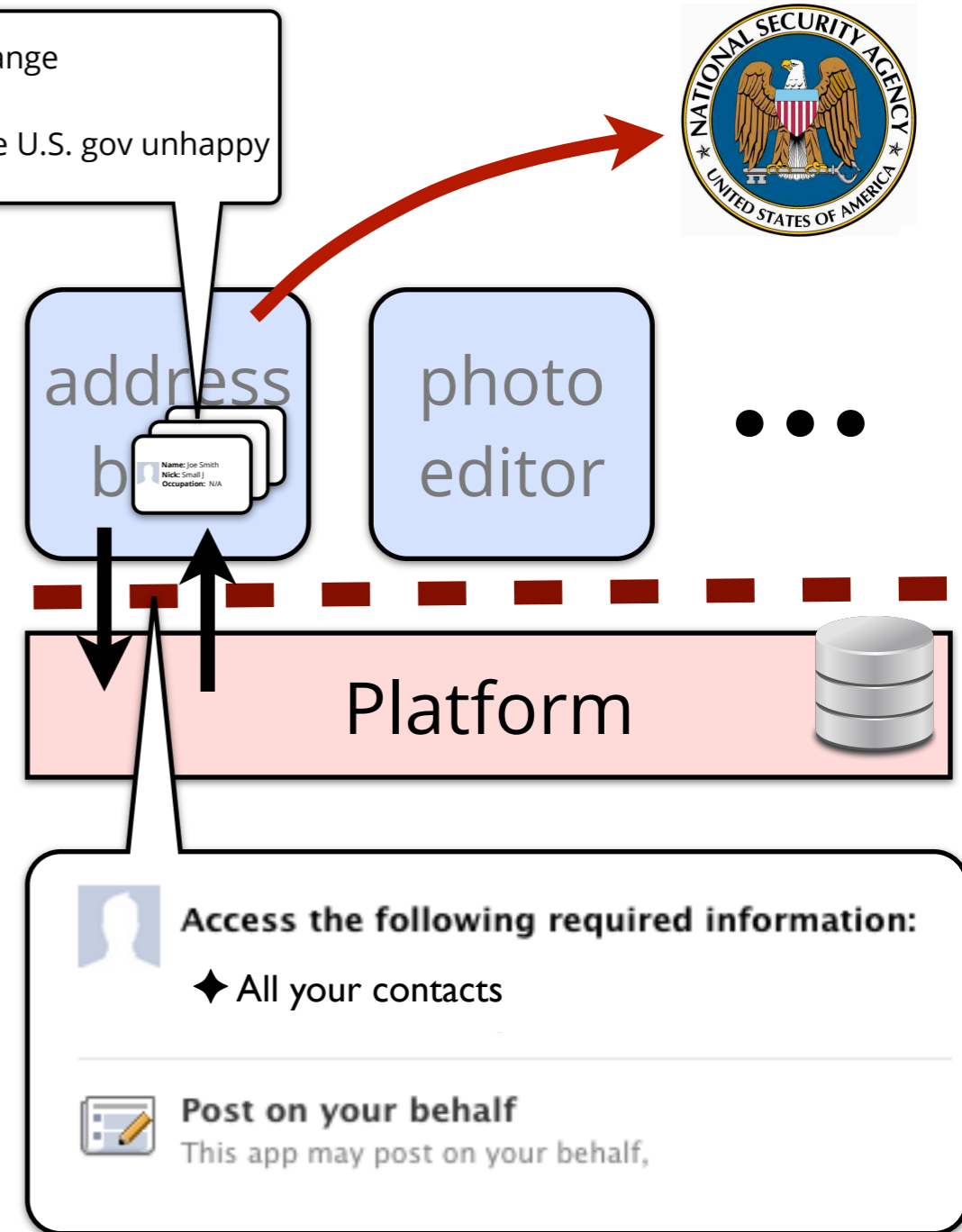
Name: Jullian Assange
Nick: Big J
Occupation: make U.S. gov unhappy



Challenge: can we ensure apps don't leak data?

Current approach: DAC

- Restrict what data app can access
- 👎 Cannot guarantee what app does with data



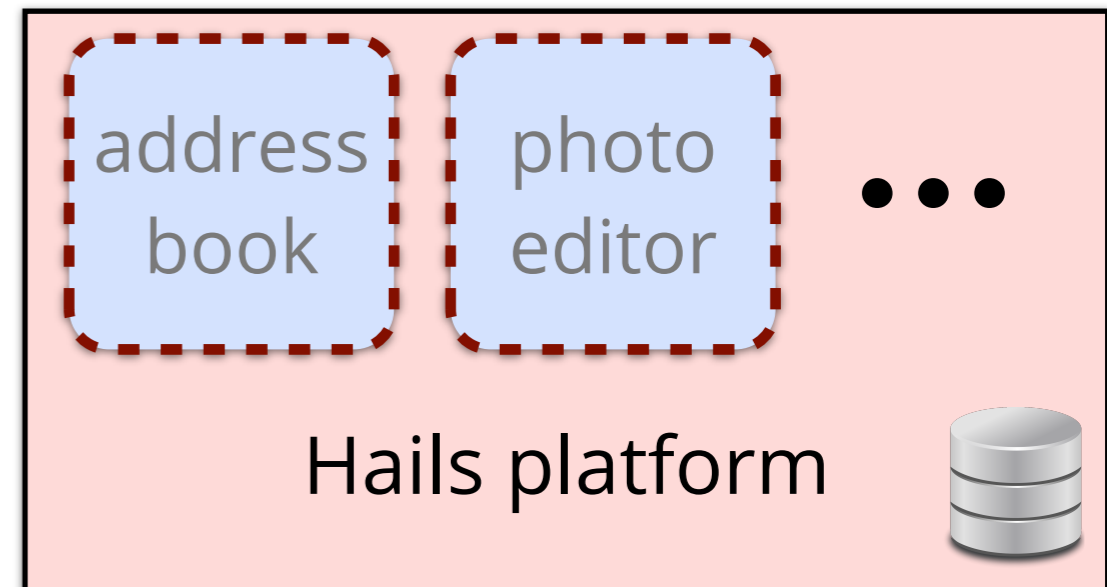


Motivation: IFC Web platforms

Solution: Information flow control Web platform: Hails

Hails IFC enforcement:

- Restrict what data app can access with clearance
- Restrict who app can communicate with depending on data it reads

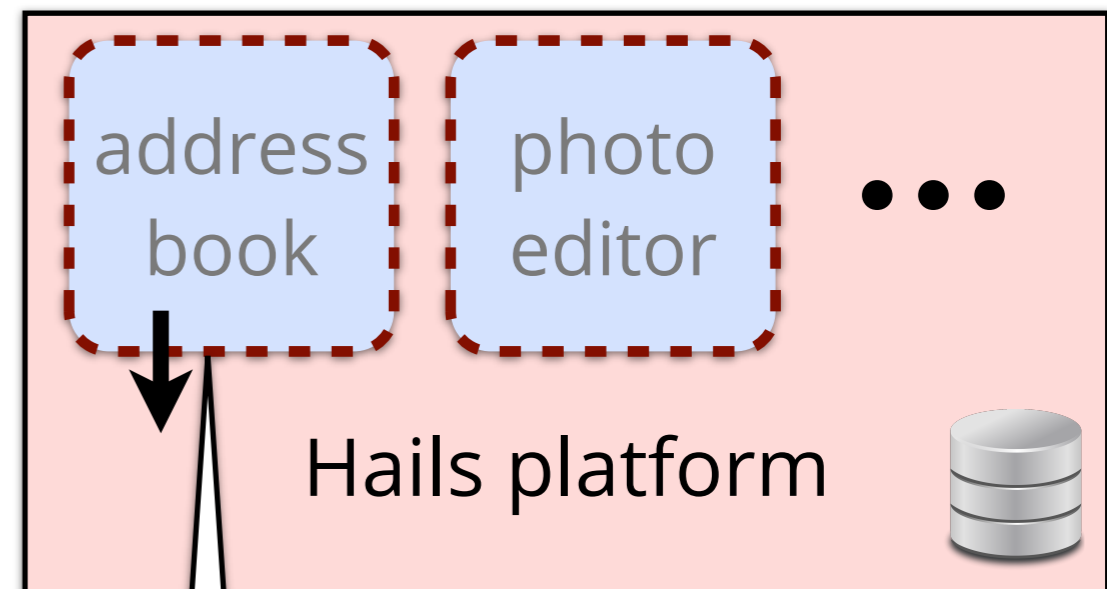


Motivation: IFC Web platforms

Solution: Information flow control Web platform: Hails

Hails IFC enforcement:

- Restrict what data app can access with clearance
- Restrict who app can communicate with depending on data it reads



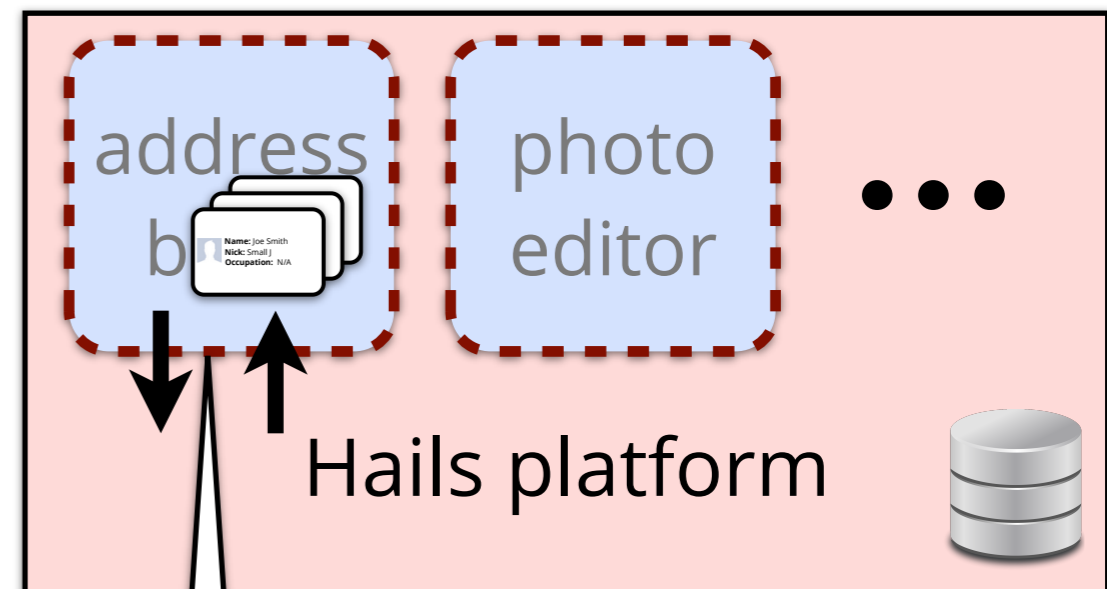
IFC: Can app read sensitive data from the database?

Motivation: IFC Web platforms

Solution: Information flow control Web platform: Hails

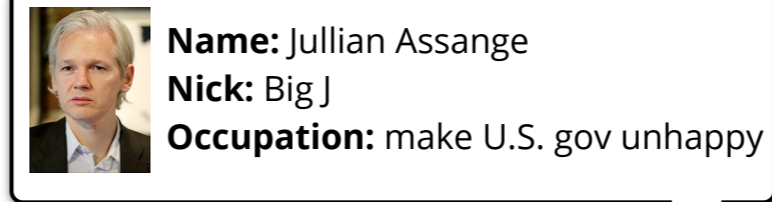
Hails IFC enforcement:

- Restrict what data app can access with clearance
- Restrict who app can communicate with depending on data it reads



IFC: Can app read sensitive data from the database?

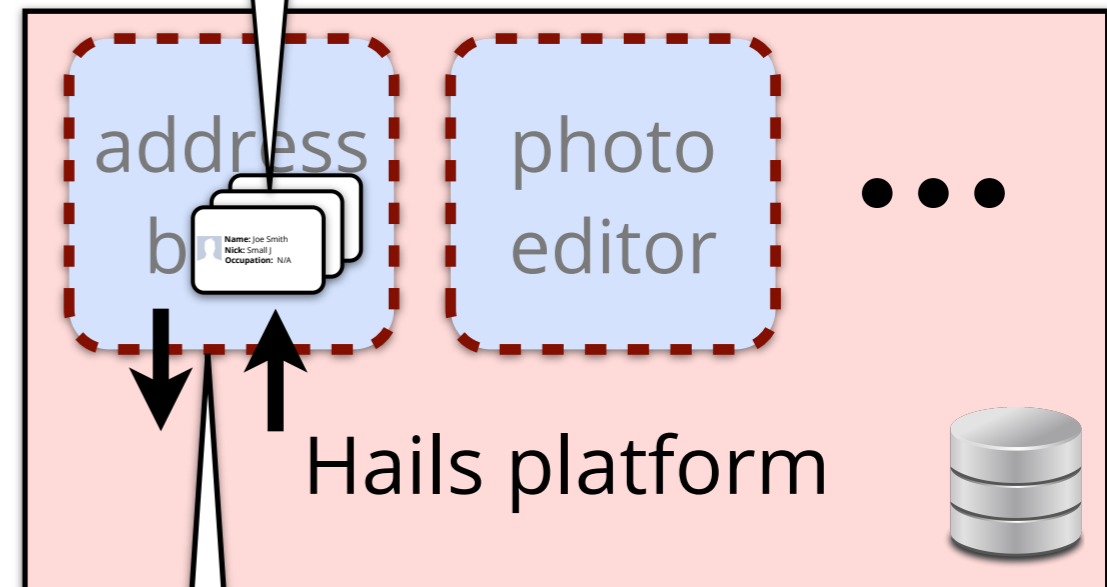
Motivation: IFC Web platforms



Solution: Information flow control Web platform: Hails

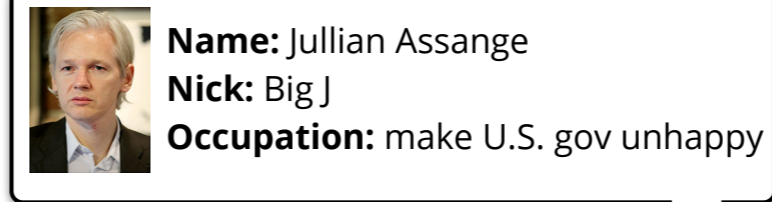
Hails IFC enforcement:

- Restrict what data app can access with clearance
- Restrict who app can communicate with depending on data it reads



IFC: Can app read sensitive data from the database?

Motivation: IFC Web platforms

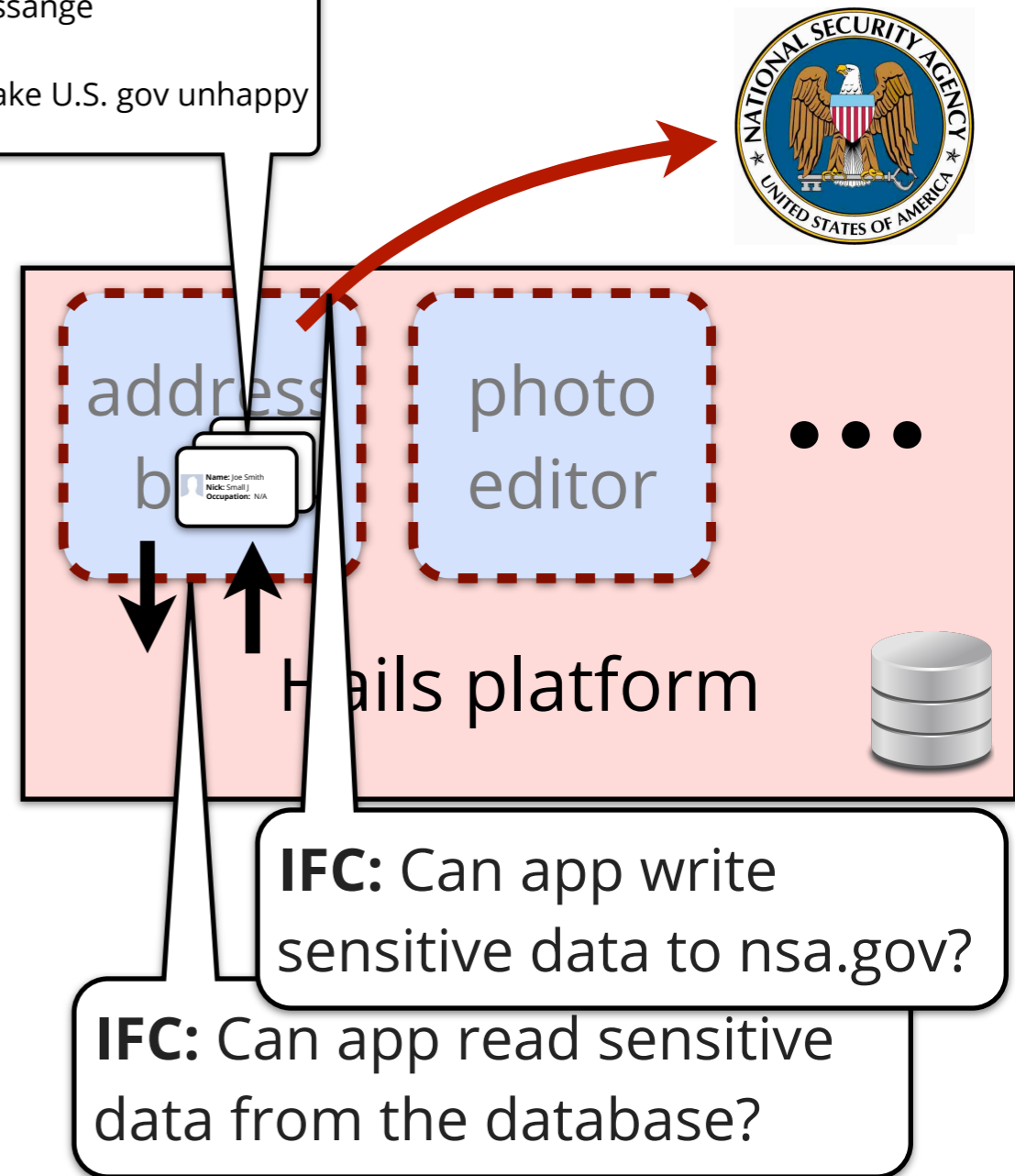


Name: Jullian Assange
Nick: Big J
Occupation: make U.S. gov unhappy

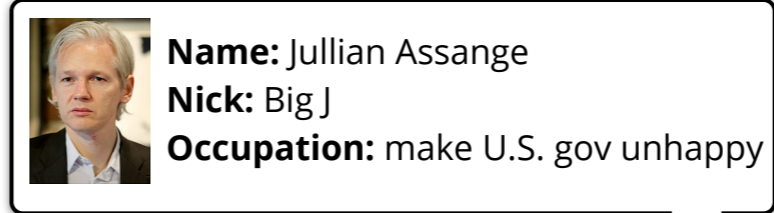
Solution: Information flow control Web platform: Hails

Hails IFC enforcement:

- Restrict what data app can access with clearance
- Restrict who app can communicate with depending on data it reads



Motivation: IFC Web platforms

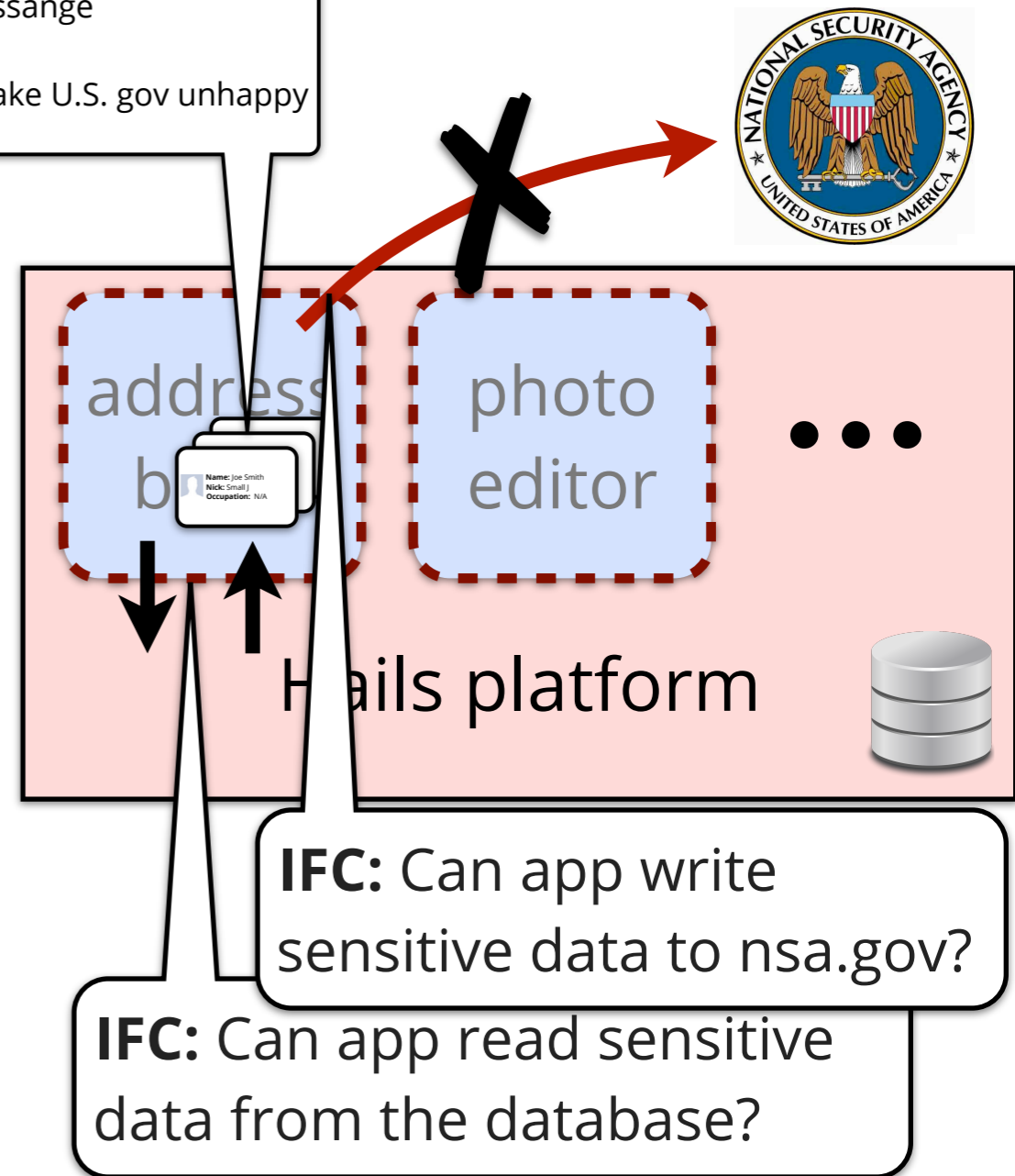


Name: Jullian Assange
Nick: Big J
Occupation: make U.S. gov unhappy

Solution: Information flow control Web platform: Hails

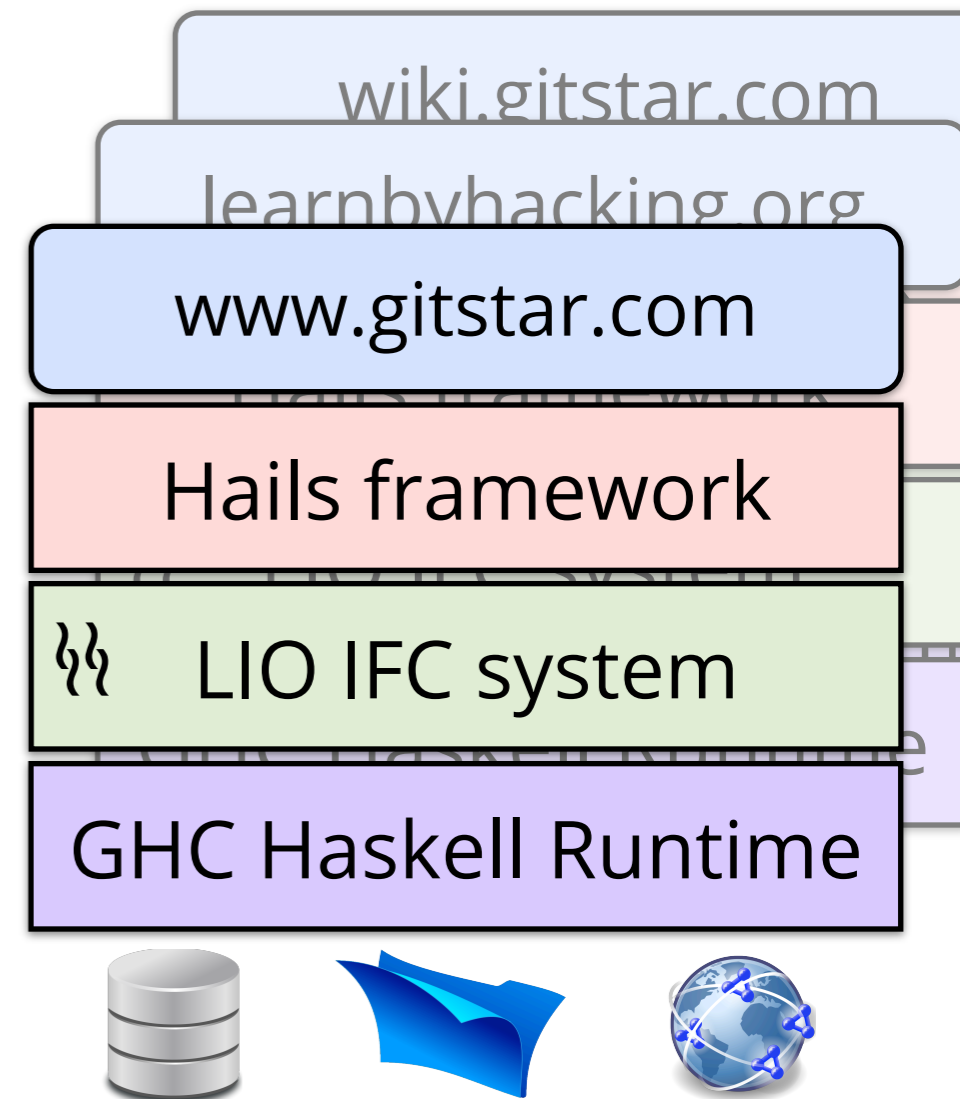
Hails IFC enforcement:

- Restrict what data app can access with clearance
- Restrict who app can communicate with depending on data it reads



Hails Web-platform framework

- Hails is built atop the LIO
 - Concurrent, dynamic, language-level IFC system
- Hails apps are LIO programs
 - Access database, filesystem, network, etc. according to IFC



Challenge: covert channels

- Malicious apps will try to leak data through any means, including covert channels
 - E.g., termination, internal timing, and external timing channels
- LIO addresses these channels at the language level

Theorem: Termination-sensitive non-interference

- *Confidentiality and integrity of data is preserved regardless of the timing/termination behavior of threads*

Challenge: covert channels


- Malicious apps will try to leak data through any means, including covert channels
 - E.g., termination, internal timing, and external timing channels
- LIO addresses these channels at the language level

Theorem: Termination sensitive non-interference

- *Confidentiality and integrity of data is preserved regardless of the timing/termination behavior of threads*

Reality check

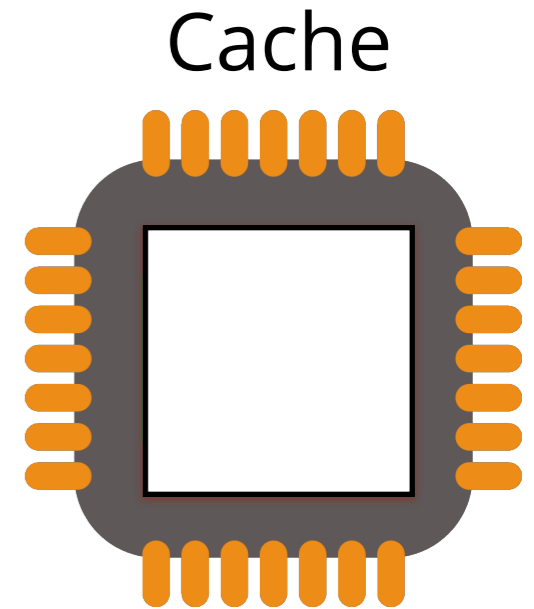
Cache Rules Everything Around Me

- Not modeling hardware features  theorem only holds for ideal execution machine
- Can usually exploit system by leveraging features not captured by model
 - E.g., finite memory, disk-head location, CPU-bus, translation look-aside buffer, L1-L3 caches

Focus: hardware-level caches

Cache-based attack

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```



Thread A

```
readArray(lowArray)
```

```
output := A
```

output
1: _____
2: _____

Thread B

```
for 1..3 do skip
```

```
output := B
```

Cache-based attack

Run 1

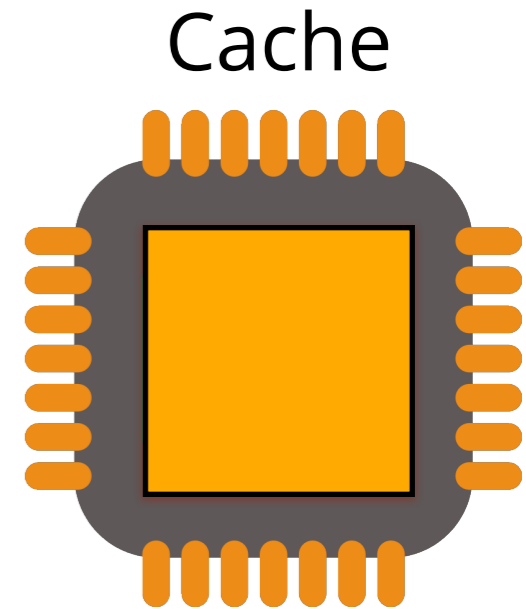
```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```

Thread B

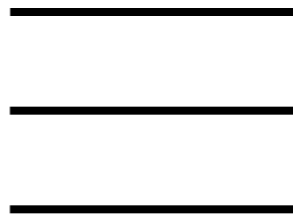
```
for 1..3 do skip
```



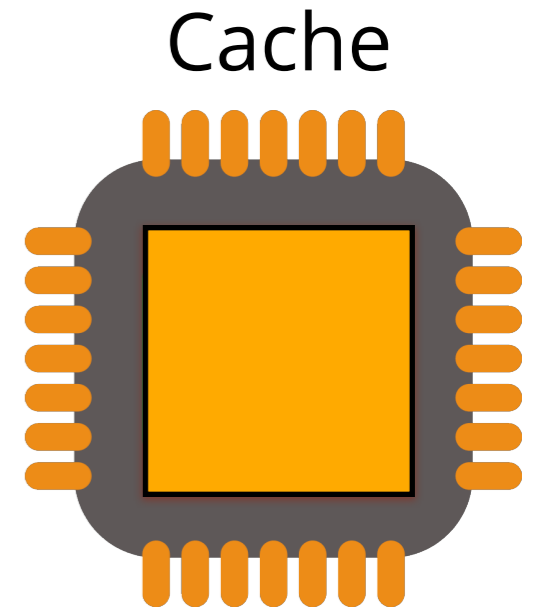
output

1:

2:



Cache-based attack



Run 1

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```



output

- 1: _____
- 2: _____

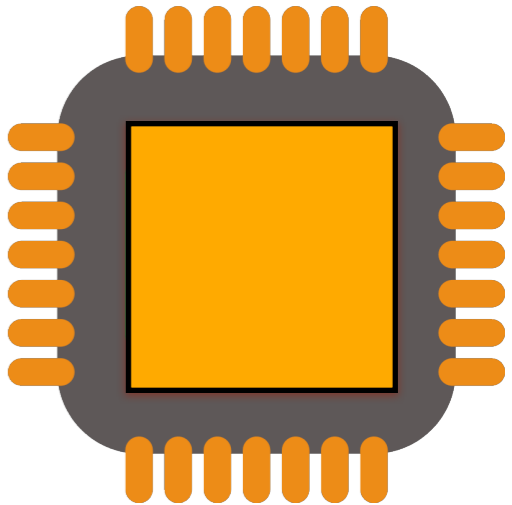
Thread B

```
for 1..3 do skip
```



Cache-based attack

Cache



Run 1

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```

```
output := A
```

Thread B

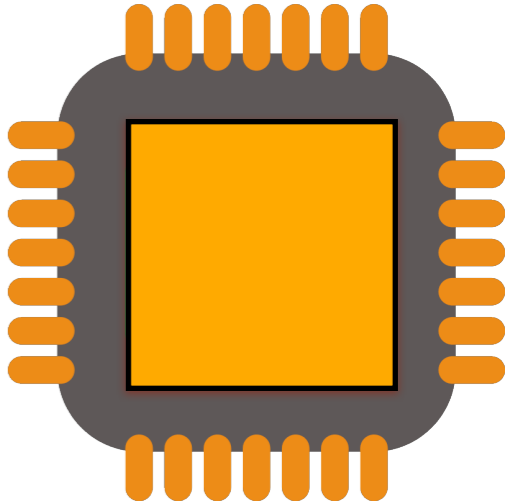
```
for 1..3 do skip
```

output

```
1: _____  
   A  
2: _____
```

Cache-based attack

Cache



Run 1

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```

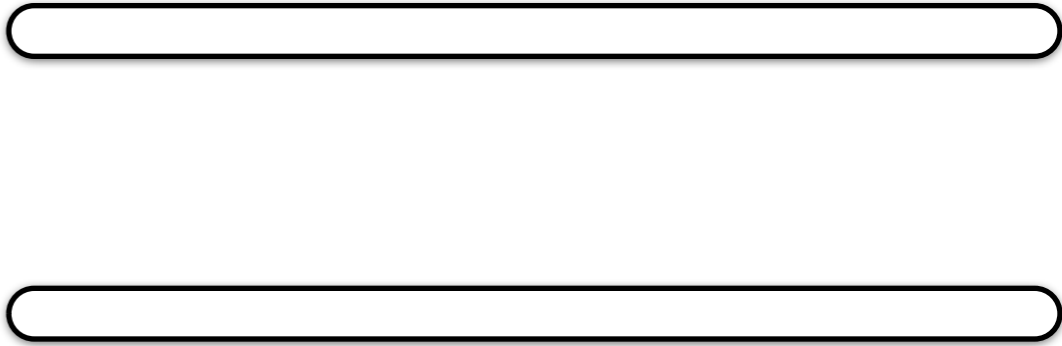
```
output := A
```

Thread B

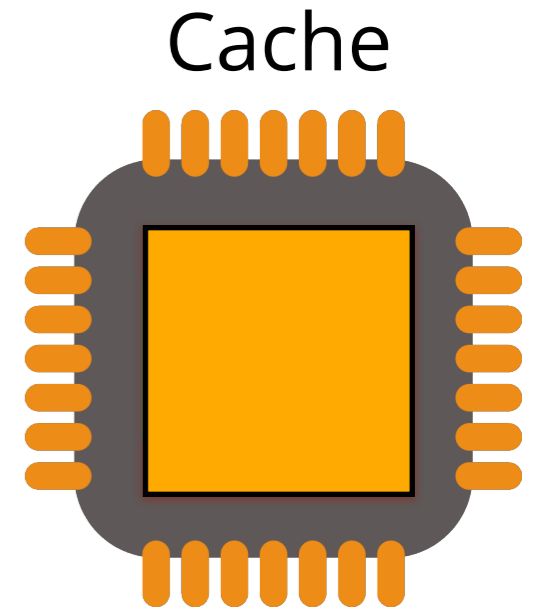
```
for 1..3 do skip
```

output

```
1: _____ A  
2: _____
```



Cache-based attack



Run 1

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```

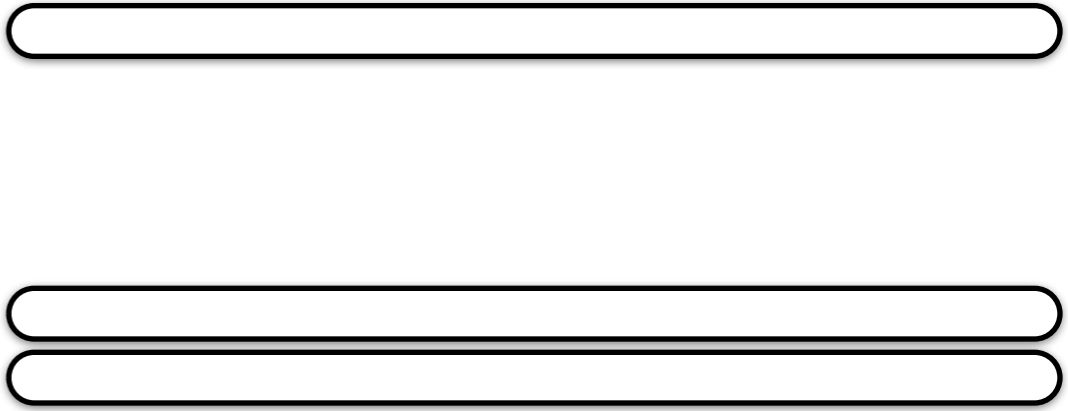
```
output := A
```

output

```
1: _____  
   A  
2: _____
```

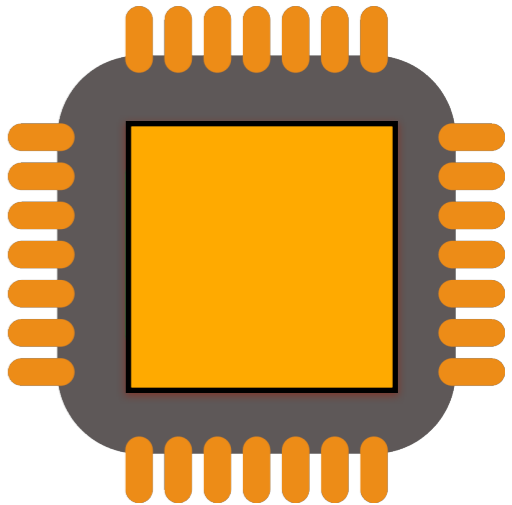
Thread B

```
for 1..3 do skip
```



Cache-based attack

Cache



Run 1

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```

```
output := A
```

Thread B

```
for 1..3 do skip
```

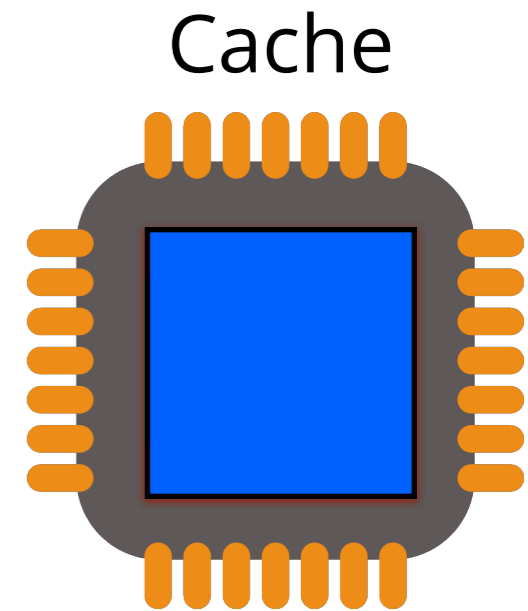
```
output := B
```

output

1: B A

2:

Cache-based attack



Run 2

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```

```
output := A
```

Thread B

```
for 1..3 do skip
```

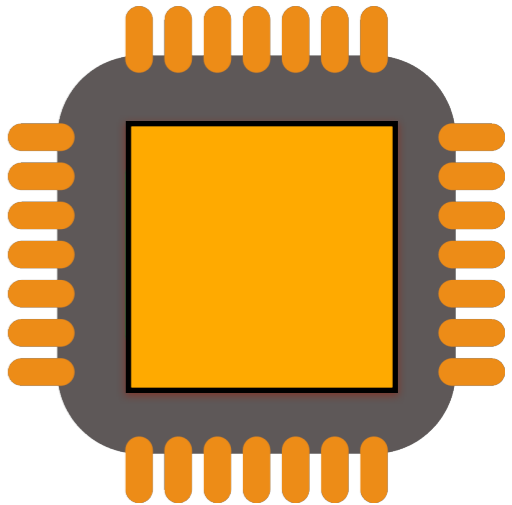
output

```
1:  B A
```

```
2:  
```

Cache-based attack

Cache



Run 2

```
lowArray := fillArray(■)
if friend == "Julian Assange"
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```

```
output := A
```

Thread B

```
for 1..3 do skip
```

```
output := B
```

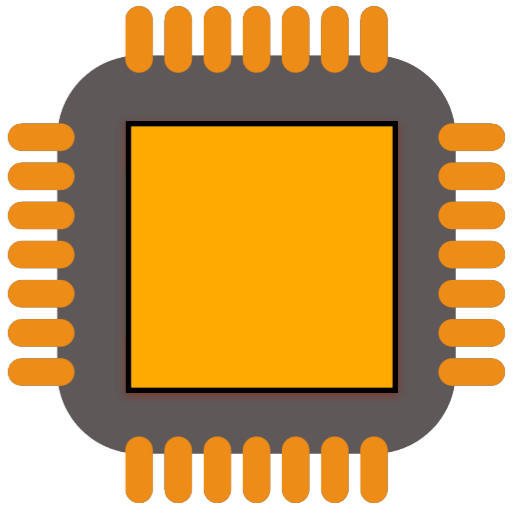
output

1: B A

2: B

Cache-based attack

Cache



Run 2

```
lowArray := fillArray(■)
if friend == "Julian Assange"
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```

```
output := A
```

```
output := A
```

Thread B

```
for 1..3 do skip
```

```
output := B
```

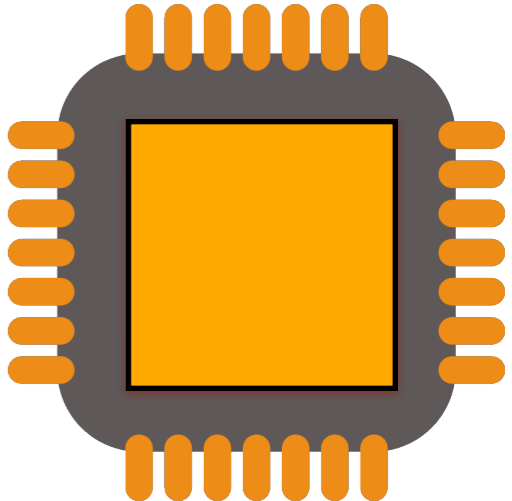
output

1: B A

2: A B

Cache-based attack

Cache



Run 2

```
lowArray := fillArray(■)
if friend == "Julian Assange"
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```

```
output := A
```

```
output := A
```

Thread B

```
for 1..3 do skip
```




```
output := B
```

output

1:	B	A
2:	A	B

Directly encodes  / 

Cache-based attack




- Reintroduction of the internal timing attack
 - Threads race to a common resource (output)
 - Can be used to leak secrets internally to app
- Trivial L1-cache attack leaks at 0.75 bits / s
 - Applicable to Hails' gitstar.com platform  leak list of collaborators on a private project in < 1 min

Outline

- Motivation: Need for Hails IFC Web platforms
- Cache-based attack on IFC platforms
- Existing countermeasures
- New countermeasure: instruction-based scheduling
 - Benefits and limitations





Countermeasures

1. Flush the cache on every context switch

-  Every thread quantum starts with fresh cache
-  Flushing the cache is prohibitively expensive for Hails user-level threads
-  Does not address resources such as CPU bus contention




Countermeasures

2. Use CPU no-fill cache mode

-  Scheduling secret threads bypasses cache, cannot affect public threads
-  Secret threads never use the cache
-  Does not scale beyond 2 security levels
-  Does not address resources such as the TLB and CPU bus contention

Countermeasures

3. Partition the cache

-  Threads at different security level effectively have a private (part of the) cache
-  Does not scale to platform with hundreds of users that come and go (current OS limit: 16)
-  Does not address resources such as the TLB and CPU bus contention

New countermeasure

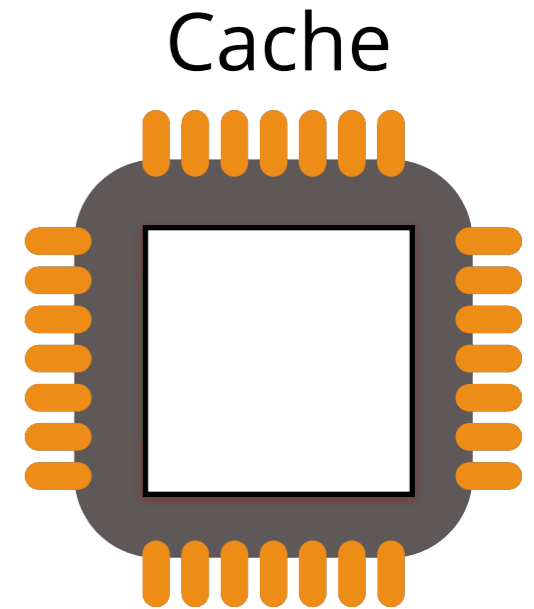
Instruction-based scheduling

Observation: The scheduling of a public thread can be affected by the timing behavior of a secret thread through the hardware cache

Solution: Schedule context switches based on number of retired instructions!

~~Cache-based attack~~

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```



Thread A

```
readArray(lowArray)
```

```
output := A
```

output
1: _____
2: _____

Thread B

```
for 1..3 do skip
```

```
output := B
```


~~Cache-based attack~~

Run 1

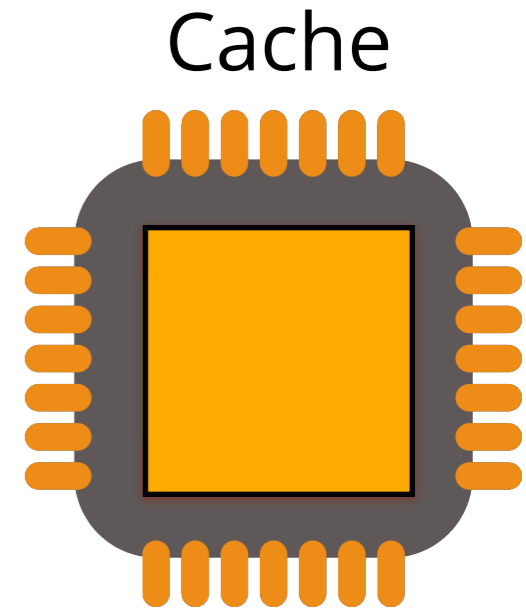
```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```

Thread B

```
for 1..3 do skip
```



output

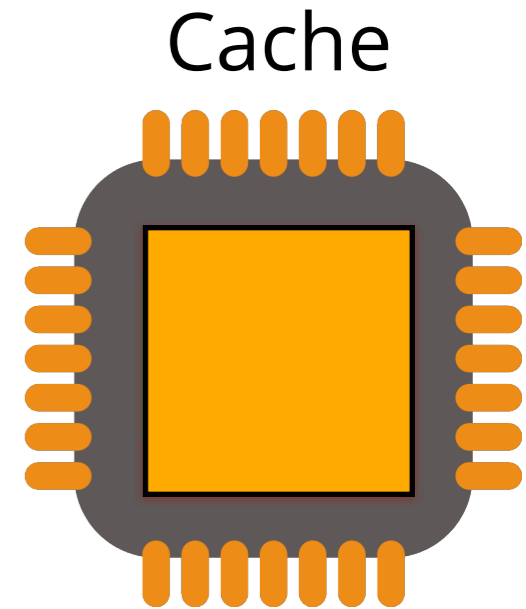
1:

2:

~~Cache-based attack~~

Run 1

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```



Thread A

readArray(lowArray)



Thread B

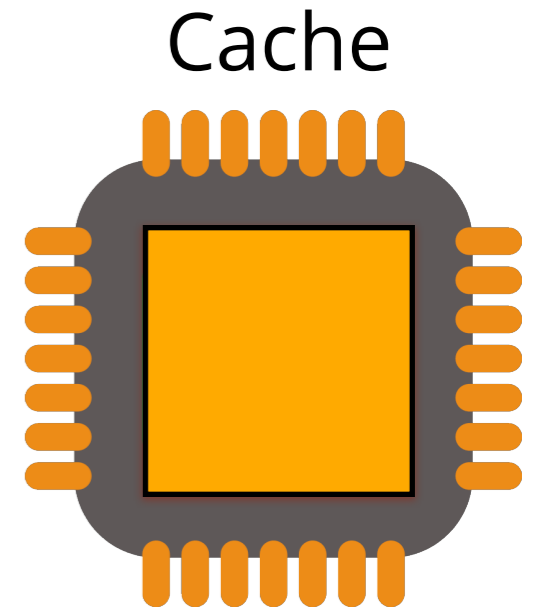
for 1..3 do skip

output

1: _____

2: _____

~~Cache-based attack~~



Run 1

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```



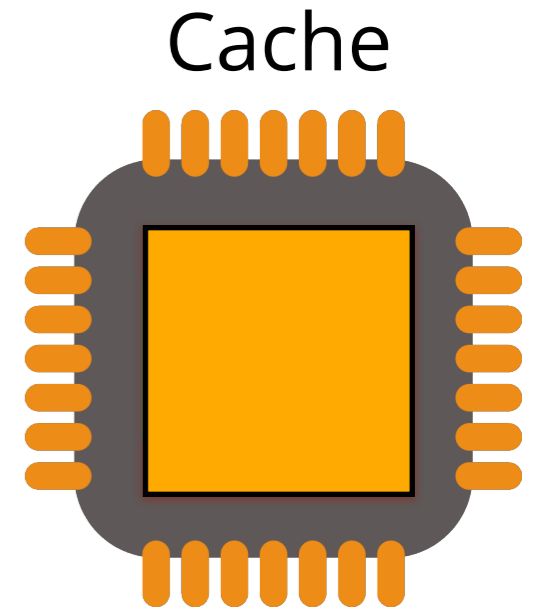
output
1: _____
2: _____

Thread B

```
for 1..3 do skip
```



~~Cache-based attack~~



Run 1

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```



```
output := A
```

output

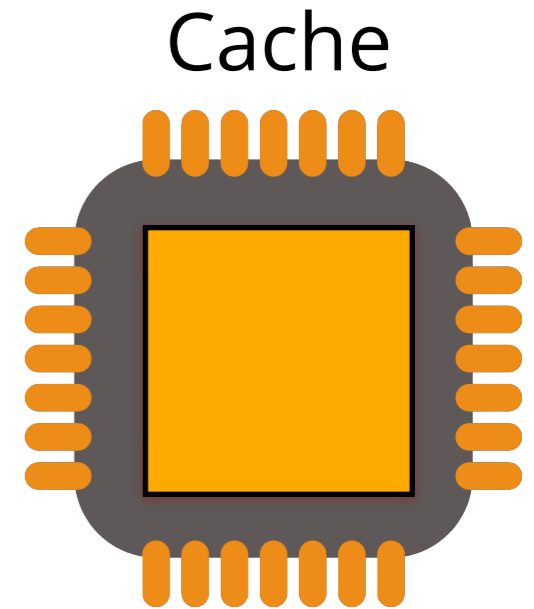
```
1: _____  
   A  
2: _____
```

Thread B

```
for 1..3 do skip
```



~~Cache-based attack~~



Run 1

```
lowArray := fillArray(■)
if friend == "Julian Assange"
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```



```
output := A
```

output

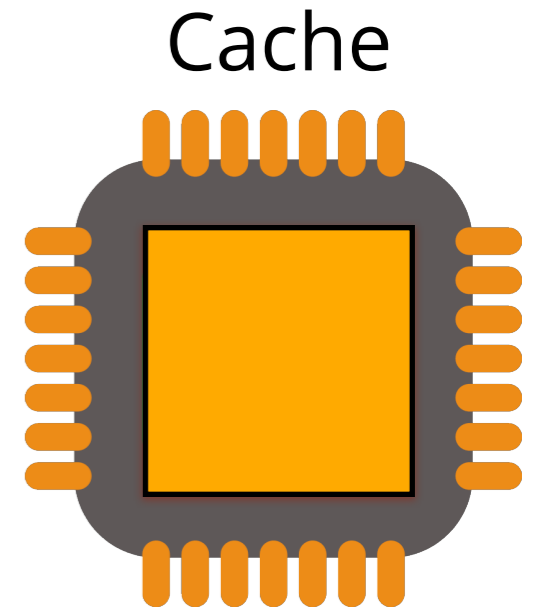
```
1: _____
   A
2: _____
```

Thread B

```
for 1..3 do skip
```



~~Cache-based attack~~



Run 1

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```



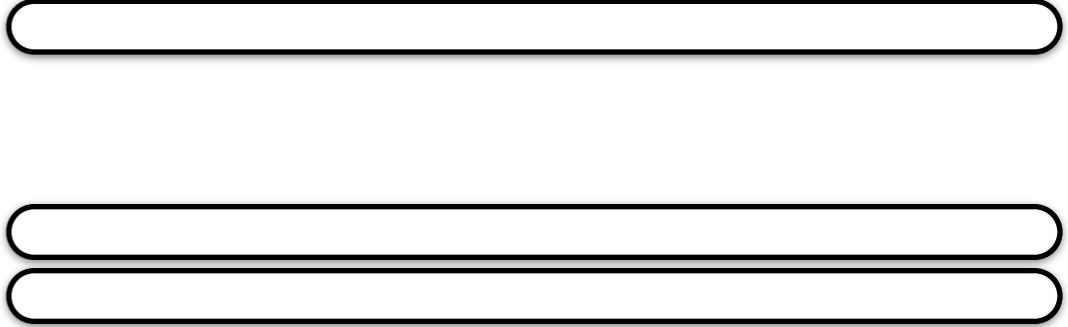
```
output := A
```

output

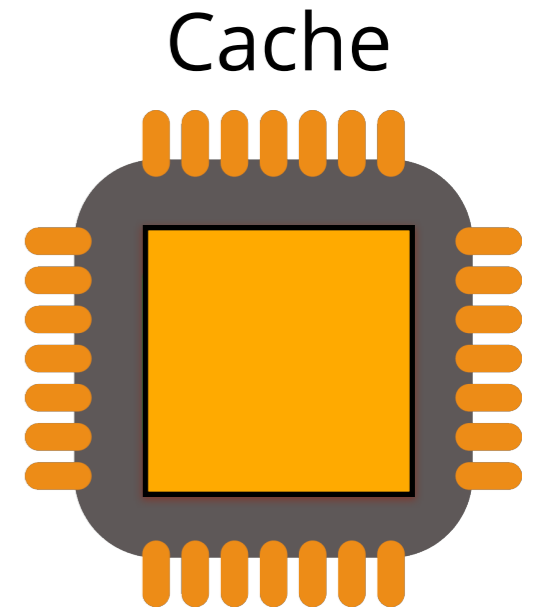
```
1: _____  
   A  
2: _____
```

Thread B

```
for 1..3 do skip
```



~~Cache-based attack~~



Run 1

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```



```
output := A
```

output

```
1:     B  A  
2:           
```

Thread B

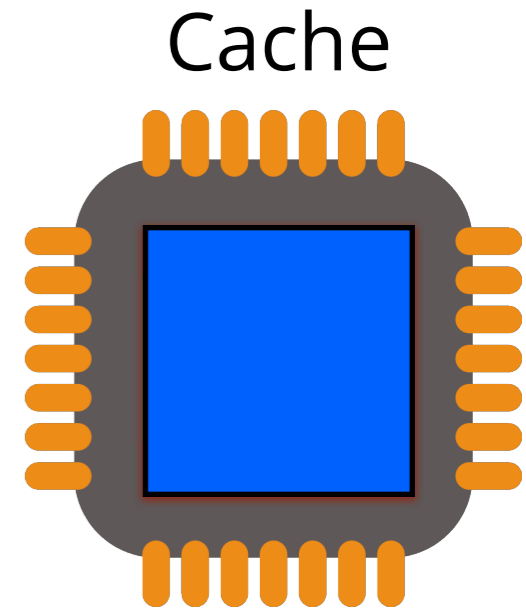
```
for 1..3 do skip
```

```
output := B
```

~~Cache-based attack~~

Run 2

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```



Thread A

```
readArray(lowArray)
```

```
output := A
```

Thread B

```
for 1..3 do skip
```

output

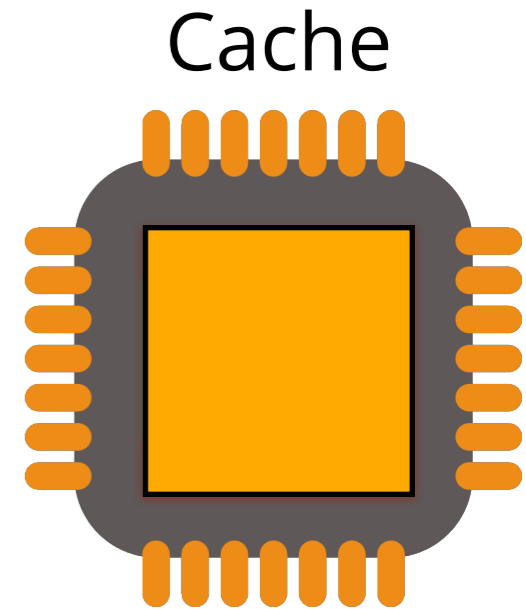
```
1:  B A
```

```
2:  
```


~~Cache-based attack~~

Run 2

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```



Thread A

```
readArray(lowArray)
```

```
output := A
```

Thread B

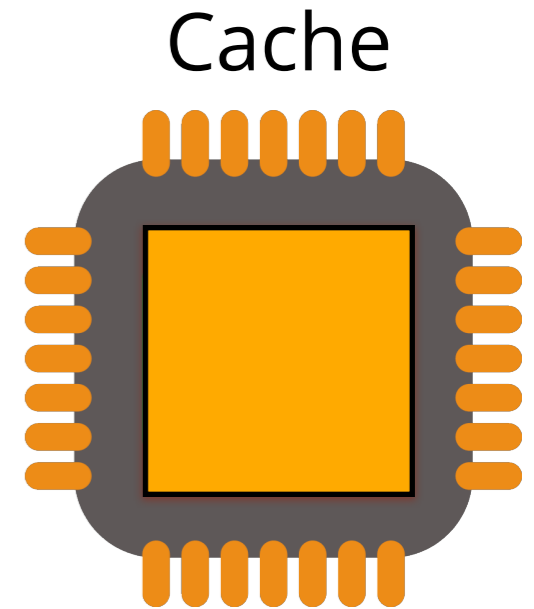
```
for 1..3 do skip
```

output

```
1:  B A
```

```
2:  
```

~~Cache-based attack~~



Run 2

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```



```
output := A
```



output

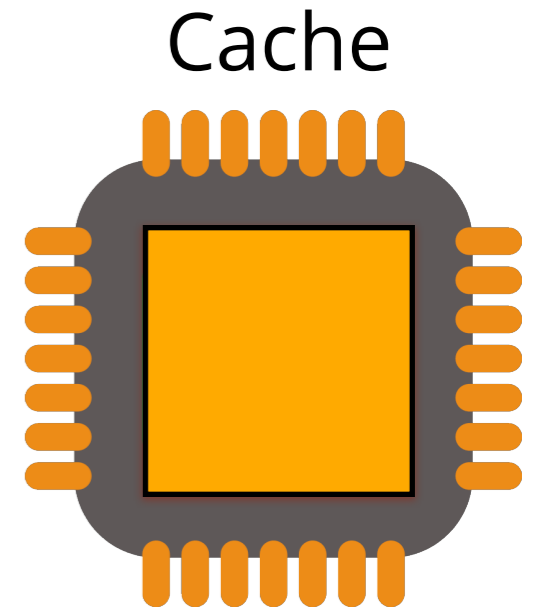
1:	B	A
2:		

Thread B

```
for 1..3 do skip
```



~~Cache-based attack~~



Run 2

```
lowArray := fillArray(■)
if friend == "Julian Assange"
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```



```
output := A
```



```
output := A
```

output

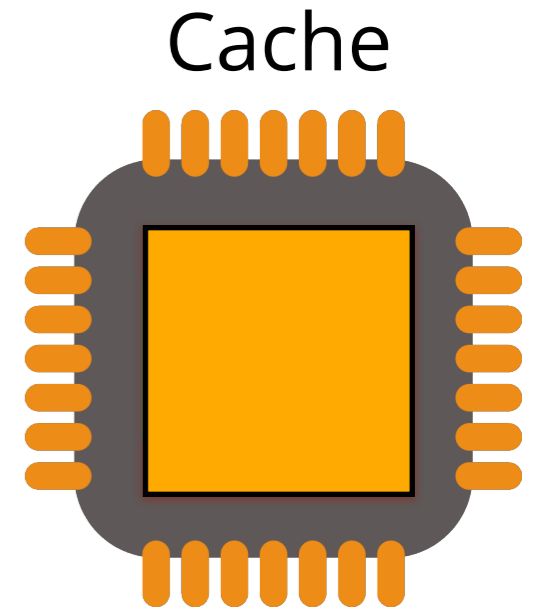
1:	B	A
2:	A	

Thread B

```
for 1..3 do skip
```



~~Cache-based attack~~



Run 2

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```



```
output := A
```



```
output := A
```

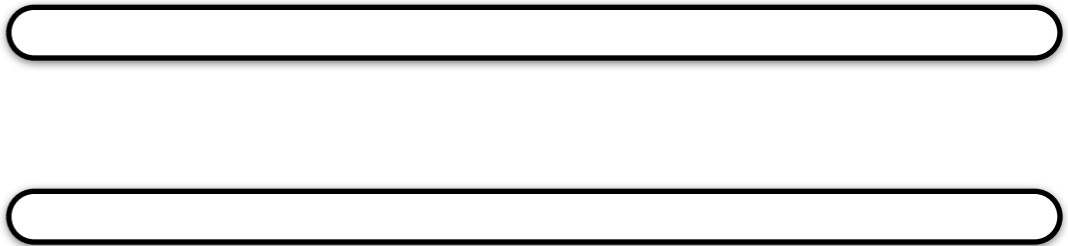
output

1: B A

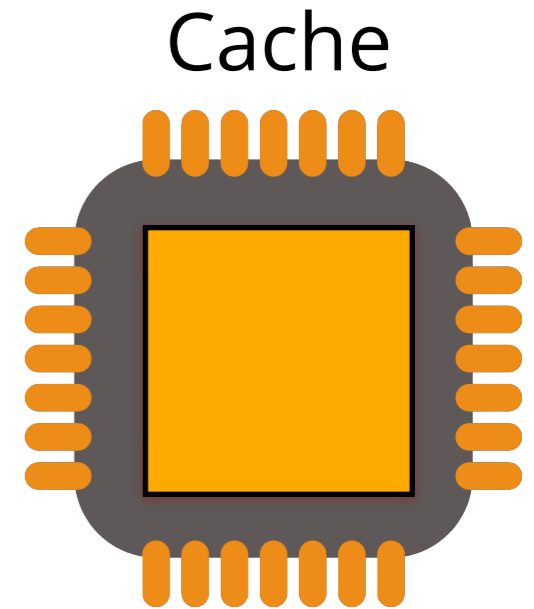
2: A

Thread B

```
for 1..3 do skip
```



~~Cache-based attack~~



Run 2

```
lowArray := fillArray(■)  
if friend == "Julian Assange"  
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```



```
output := A
```



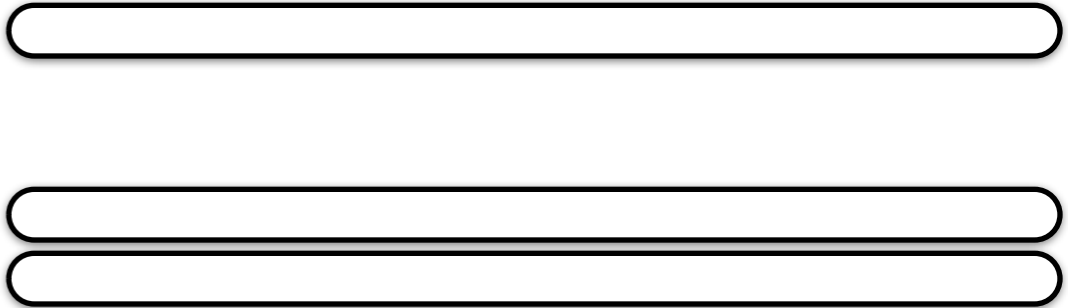
```
output := A
```

output

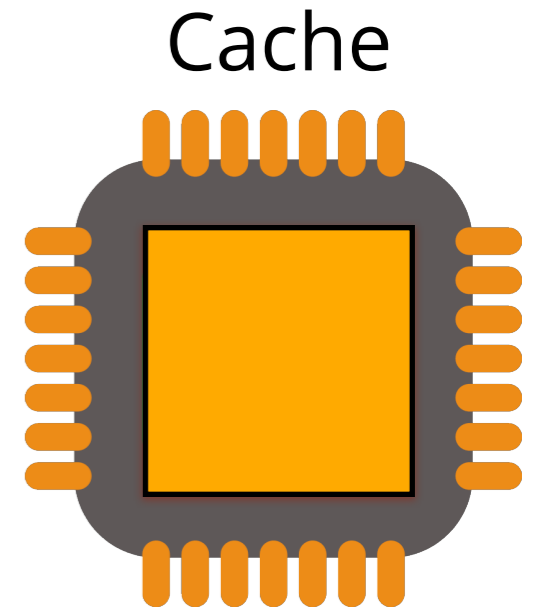
1:	B	A
2:		A

Thread B

```
for 1..3 do skip
```



~~Cache-based attack~~



Run 2

```
lowArray := fillArray(■)
if friend == "Julian Assange"
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```

```
output := A
```

```
output := A
```

output

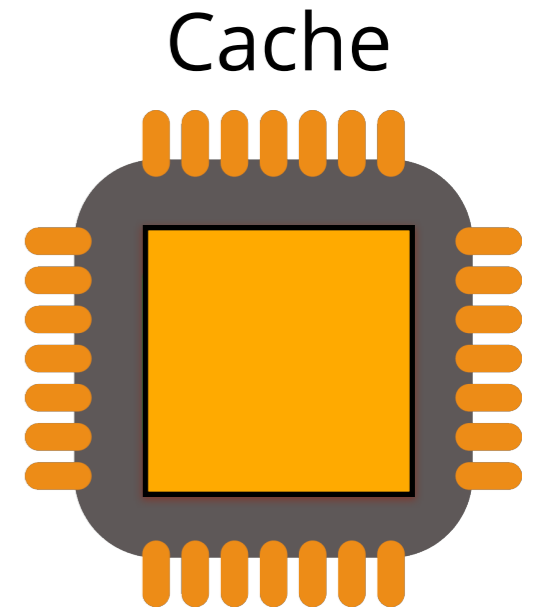
1:	B	A
2:	B	A

Thread B

```
for 1..3 do skip
```

```
output := B
```

Cache-based attack



Run 2

```
lowArray := fillArray(■)
if friend == "Julian Assange"
  highArray := fillArray(■)
```

Thread A

```
readArray(lowArray)
```

```
output := A
```

```
output := A
```

No longer encodes  / 

output



1:	B	A
2:	B	A

Thread B

```
for 1..3 do skip
```

```
output := B
```

Take away

- Secret threads can affect the duration of instructions in public threads
 - Context switching according to amount of elapsed time  can introduce public races!
- Secret threads cannot affect the number of (or which) instructions a public thread retires
 - Context switching according to number of instructions retired  no race!

Implementation

Strawman: Instruction \equiv language-level atom

- Simple to prototype, no runtime modification
- Incurs at least 10x slowdown + termination attack

Approach: Measure number of retired instructions

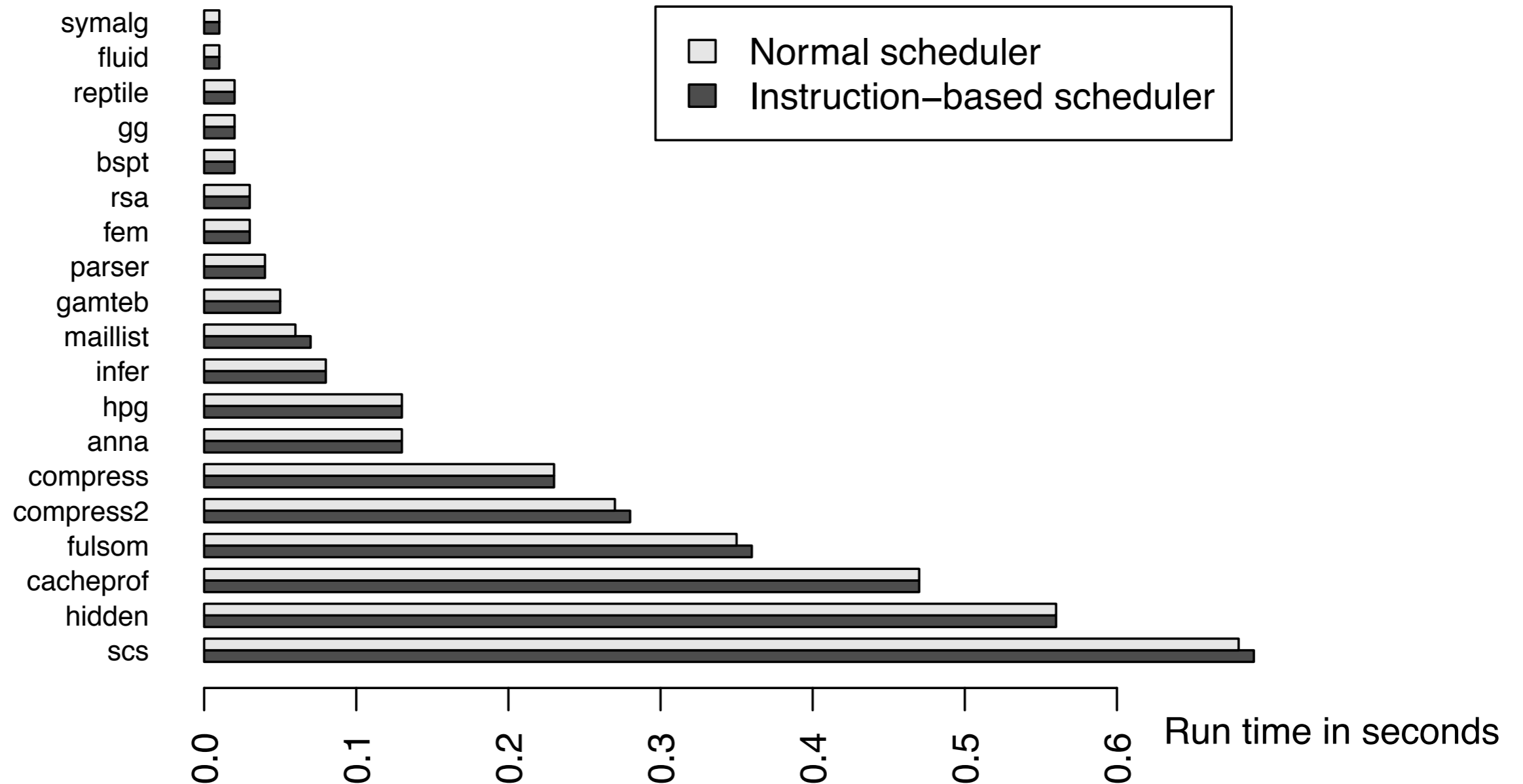
- Use hardware performance units (PMUs), readily available on modern Intel and AMD CPUs

Implementation

Replaced GHC's time-based scheduler

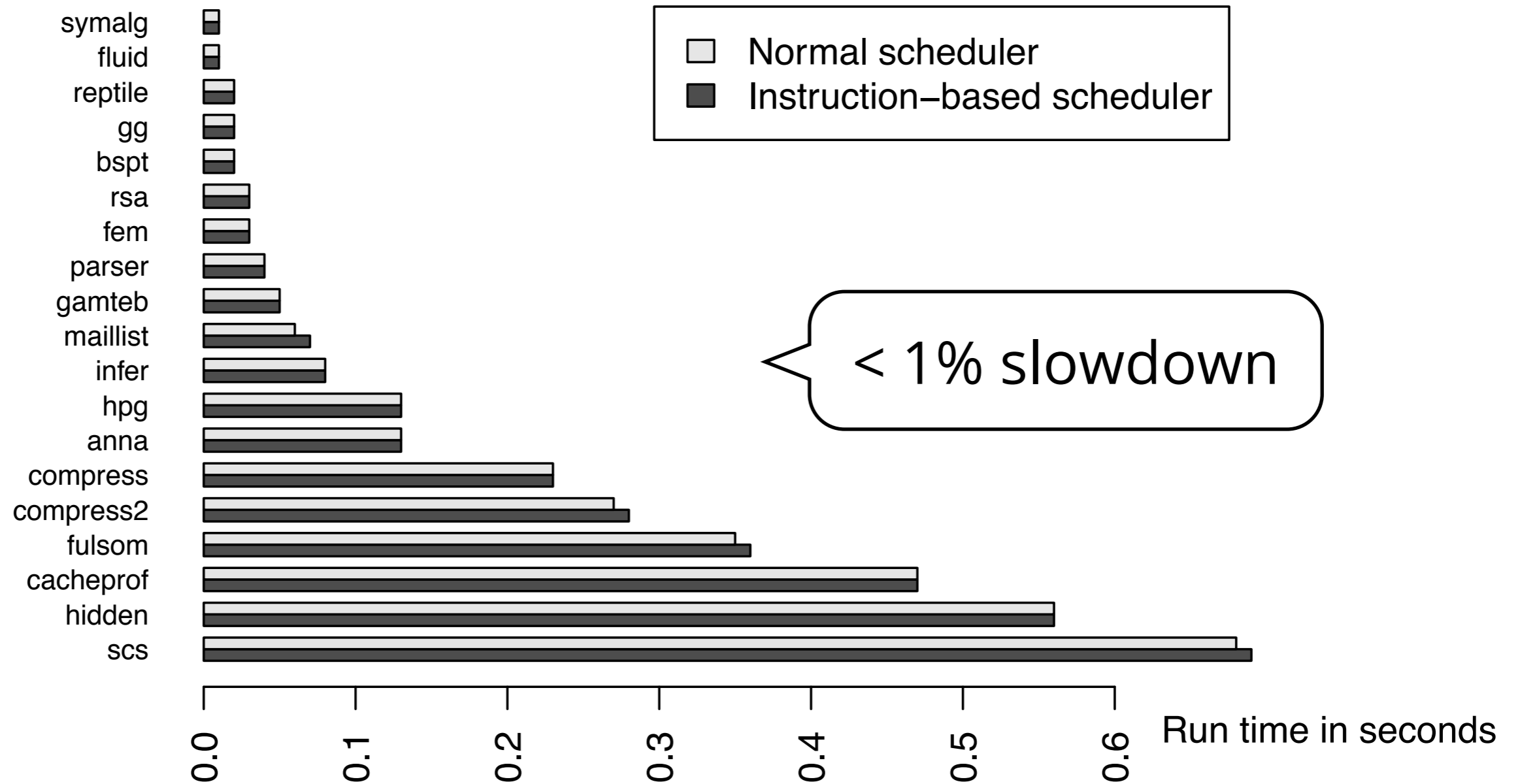
- Signal from PMU is used to context switch thread (unless the thread is not in a safe point)
- To ensure safe points are reached often, we added safe-points on every function entry
- Reset counters when thread yields to do IO

Performance impact



Disclaimer: Need code that is used in the find an instruction budget that leads to context switches at roughly 10ms intervals

Performance impact



Disclaimer: Need code that is used in the find an instruction budget that leads to context switches at roughly 10ms intervals

Conclusions

Instruction-based scheduling

- 👍 Eliminates hardware-based internal timing attacks
 - L1-L3 caches, TLB, CPU bus contention, etc.
- 👍 Scales to arbitrary number of security levels
- 👍 Almost no impact on performance
- 👎 Does not directly scale to multiple CPU cores
 - Not a big concern in network-balanced web apps