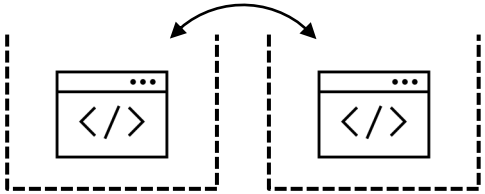
Lord of the Ring(s): Side Channel Attacks on the CPU On-Chip Ring Interconnect Are Practical

Riccardo Paccagnella, Licheng Luo, Christopher W. Fletcher

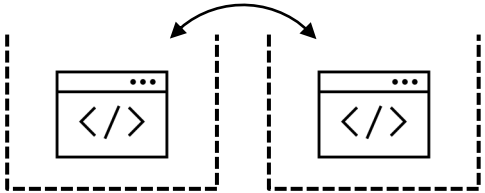
ILLINOIS

Side Channel Attacks

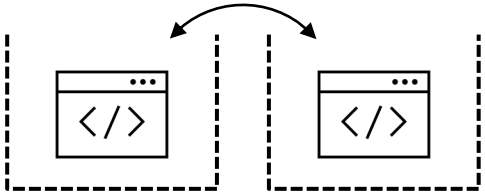
Side Channel Attacks



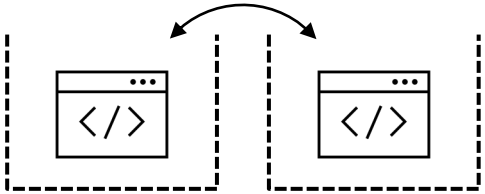
Side Channel Attacks



Side Channel Attacks



Side Channel Attacks



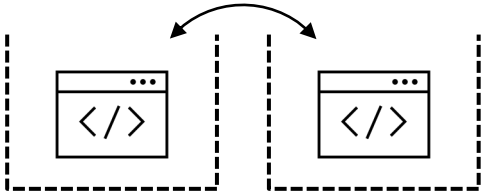
{* SECURITY *}

Google emits data-leaking proof-of-concept Spectre exploit for Intel CPUs to really get everyone's attention

I don't believe it, I had to see it, I came back, I came back haunted

Thomas Claburn in San Francisco Fri 12 Mar 2021 // 21:28 UTC SHARE

Side Channel Attacks



{* SECURITY *}

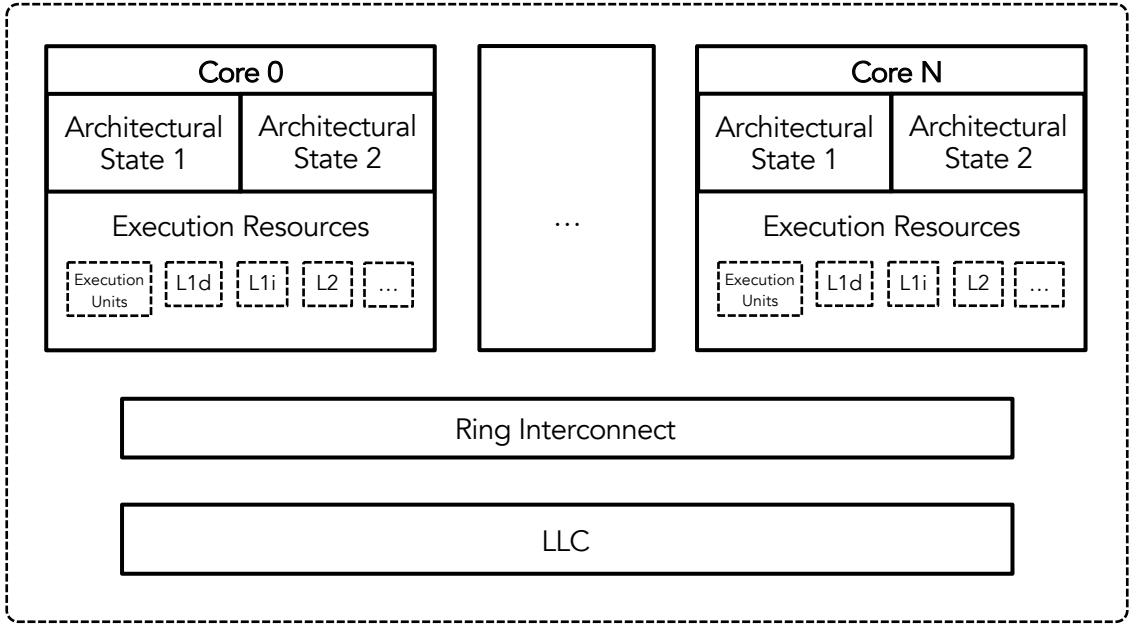
Google emits data-leaking proof-of-concept Spectre exploit for Intel CPUs to really get everyone's attention

I don't believe it, I had to see it, I came back, I came back haunted

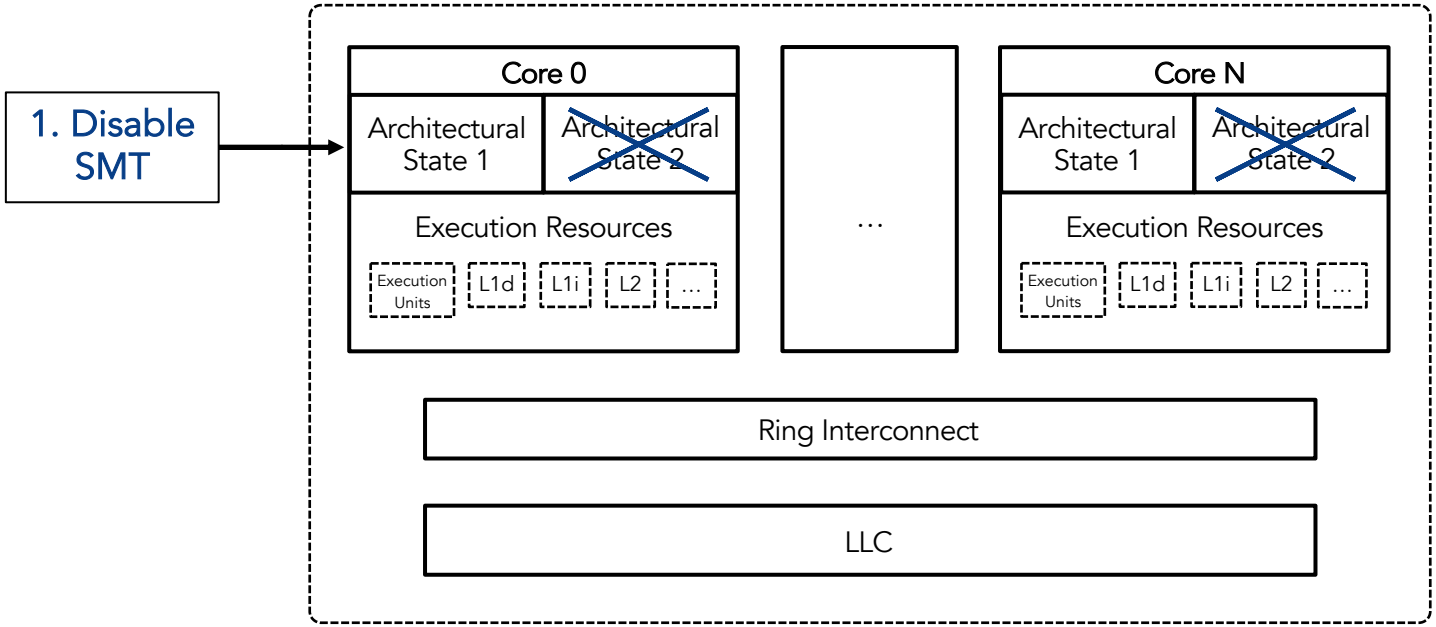
Thomas Claburn in San Francisco Fri 12 Mar 2021 // 21:28 UTC SHARE

Can we block side channels?

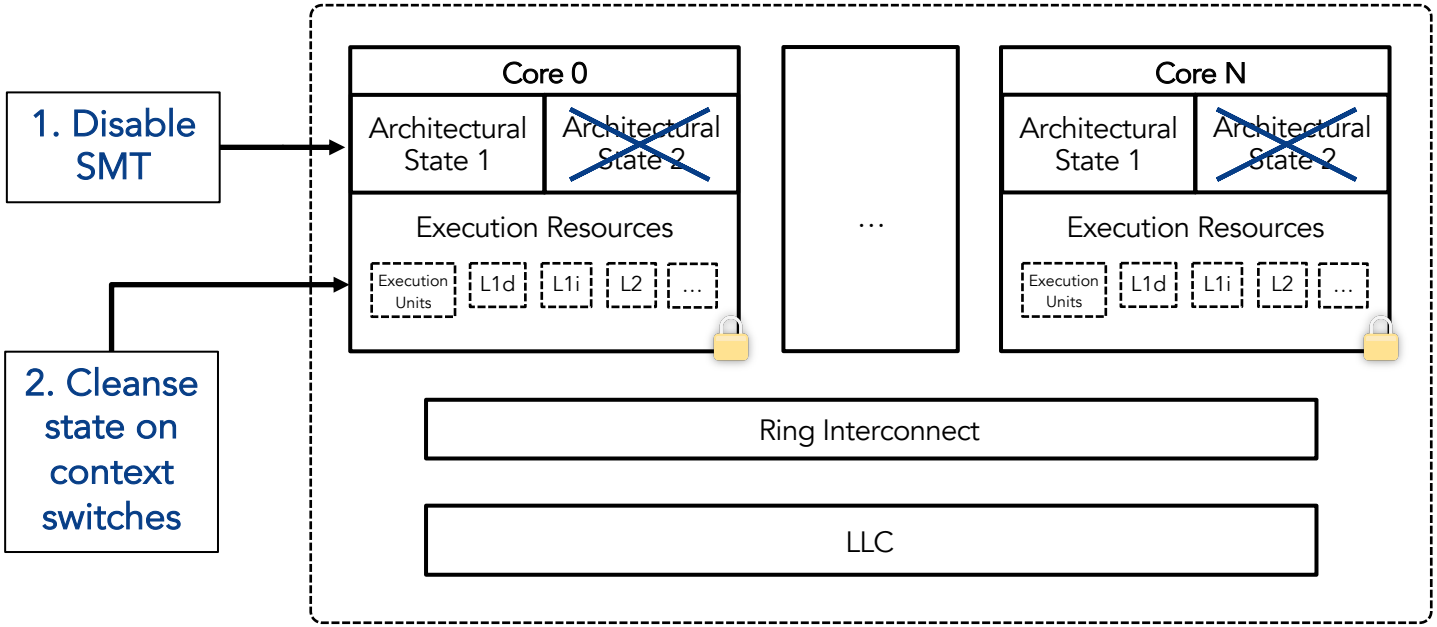
Can we block side channels?



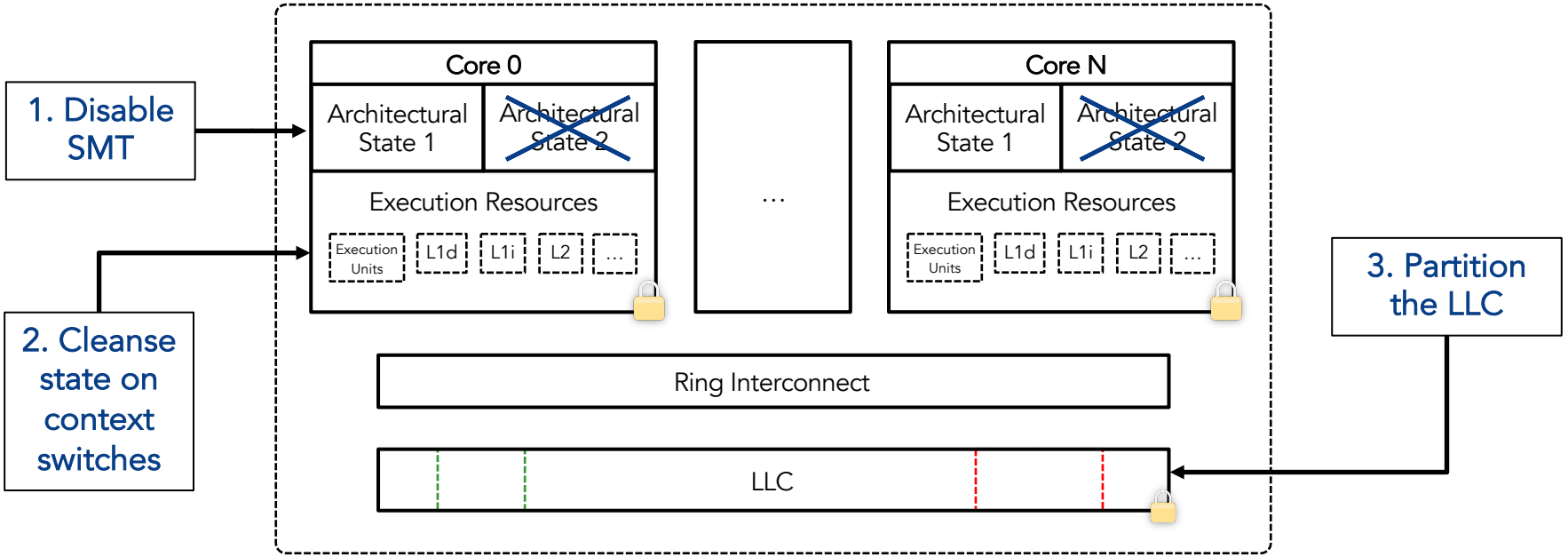
Can we block side channels?



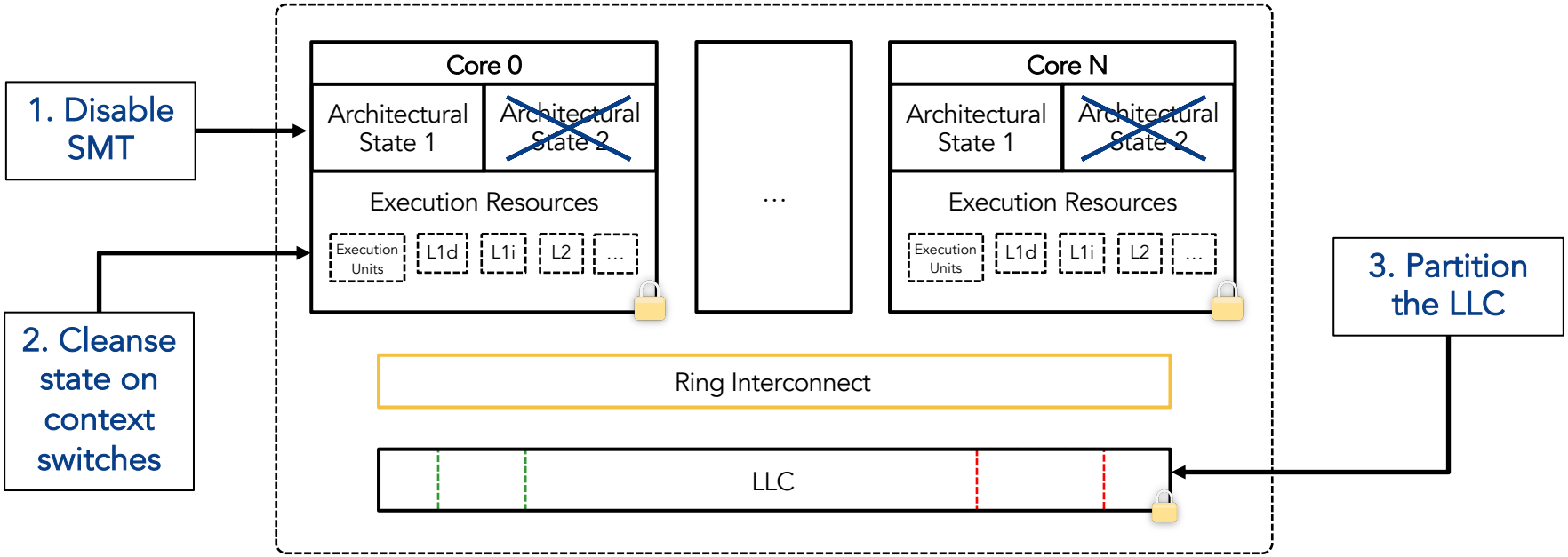
Can we block side channels?



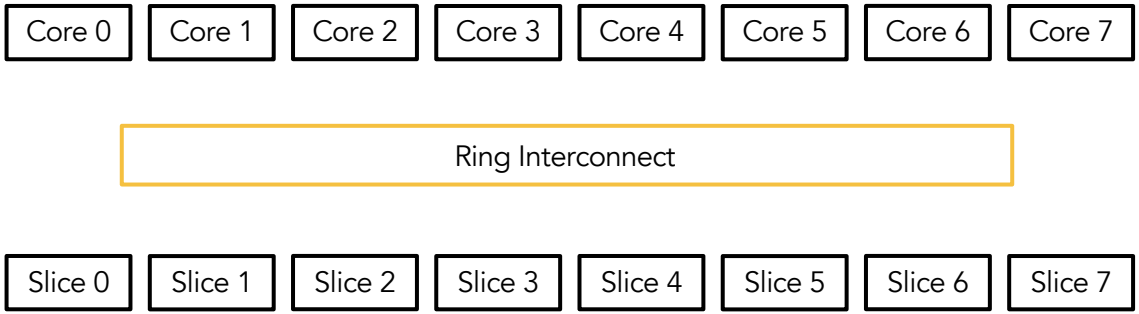
Can we block side channels?



What about the ring interconnect?

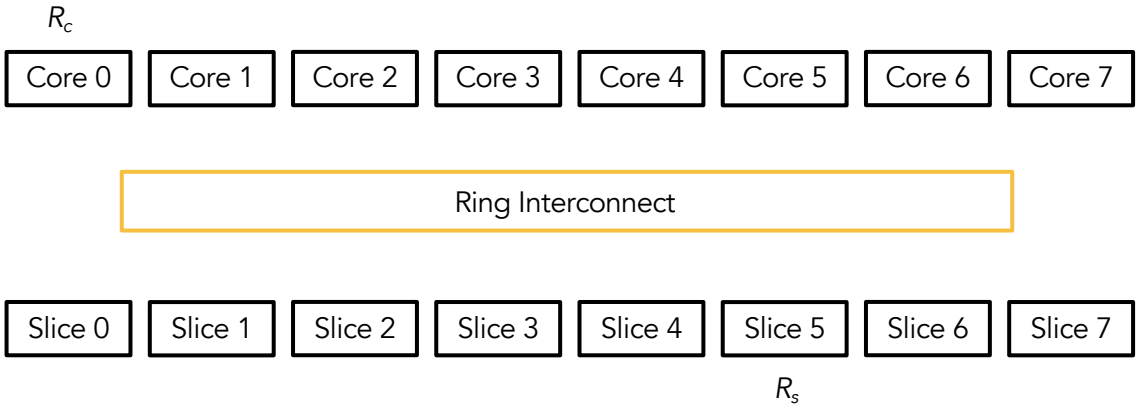


Methodology



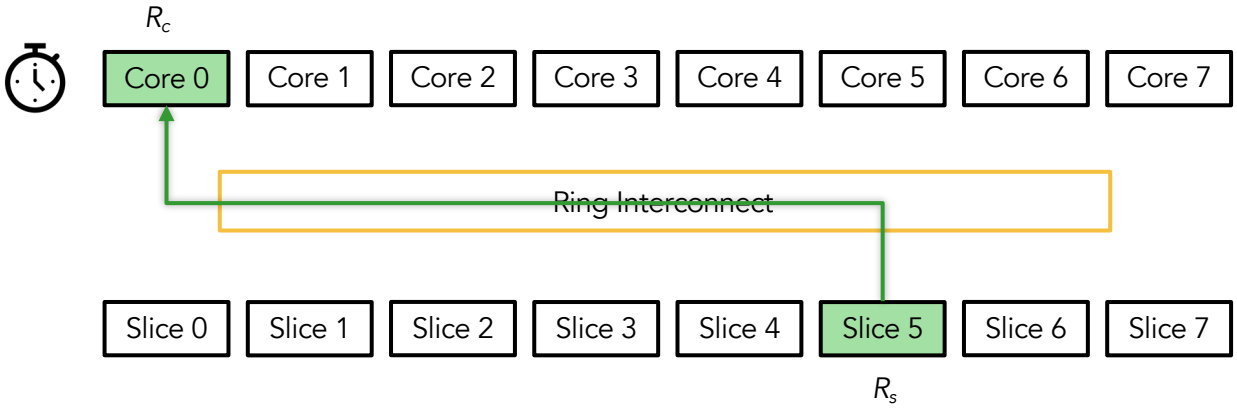
Methodology

- R_c = receiver core
- R_s = receiver LLC slice



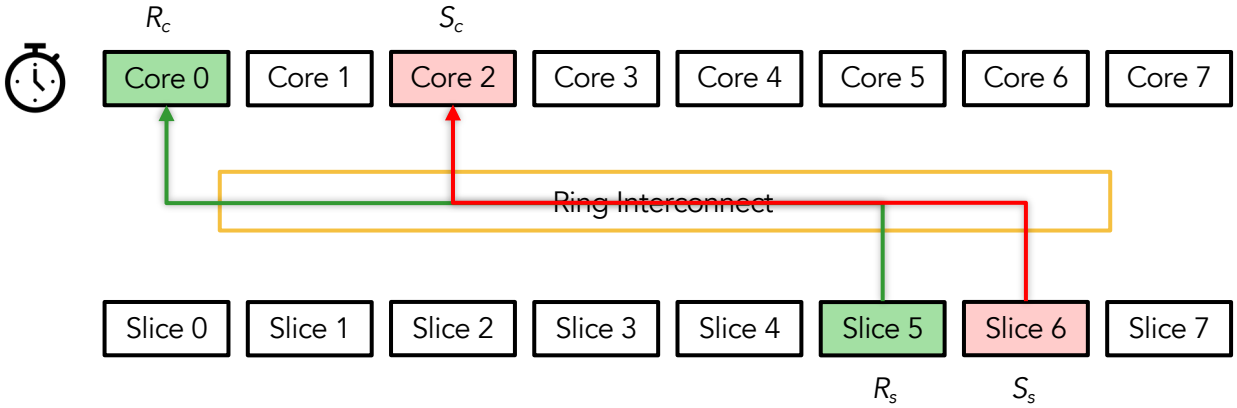
Methodology

- R_c = receiver core
- R_s = receiver LLC slice



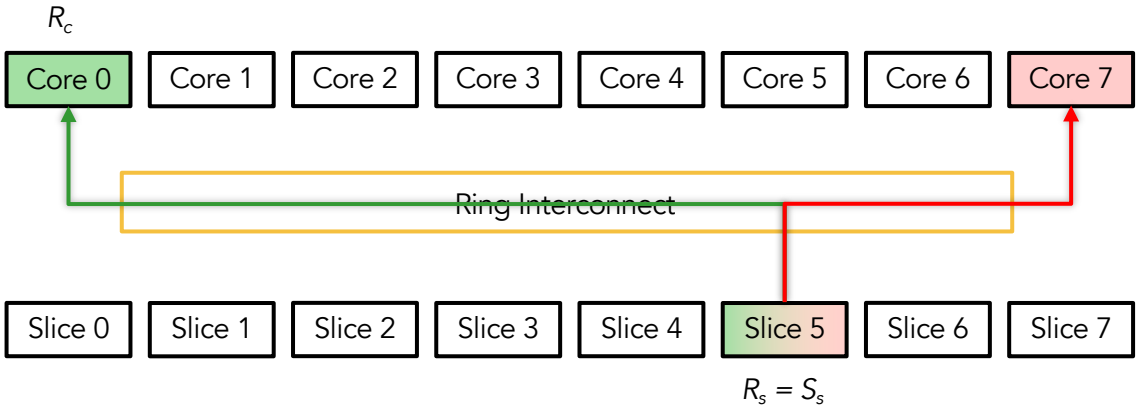
Methodology

- R_c = receiver core
- R_s = receiver LLC slice
- S_c = sender core
- S_s = sender LLC slice



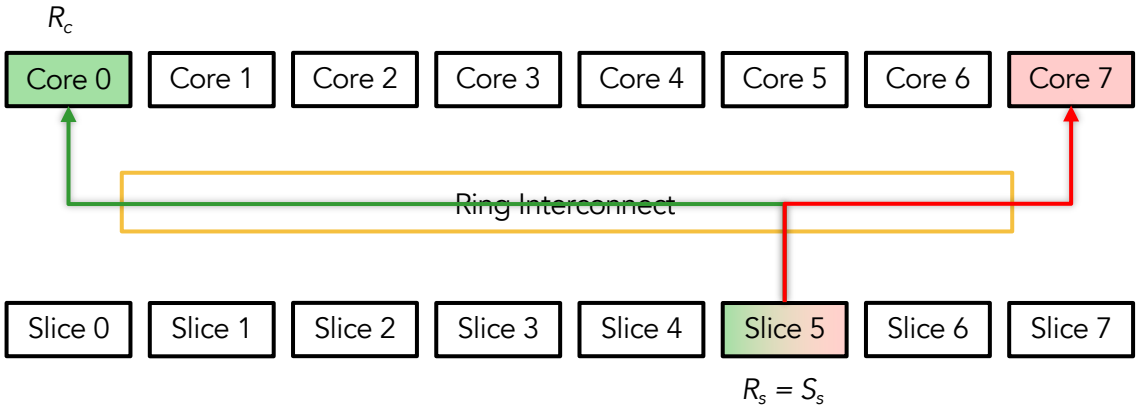
Slice Contention

- $S_s = R_s$: contention



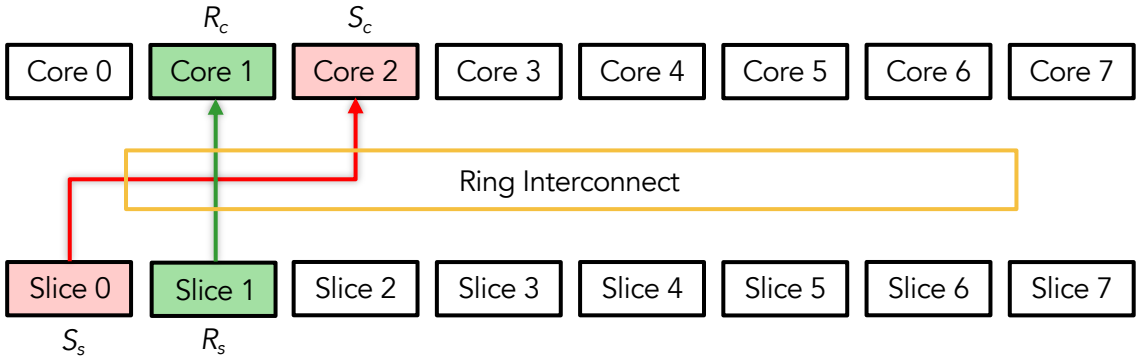
Slice Contention

- $S_s = R_s$: contention
  - Slice's request queue.
  - Slice port.



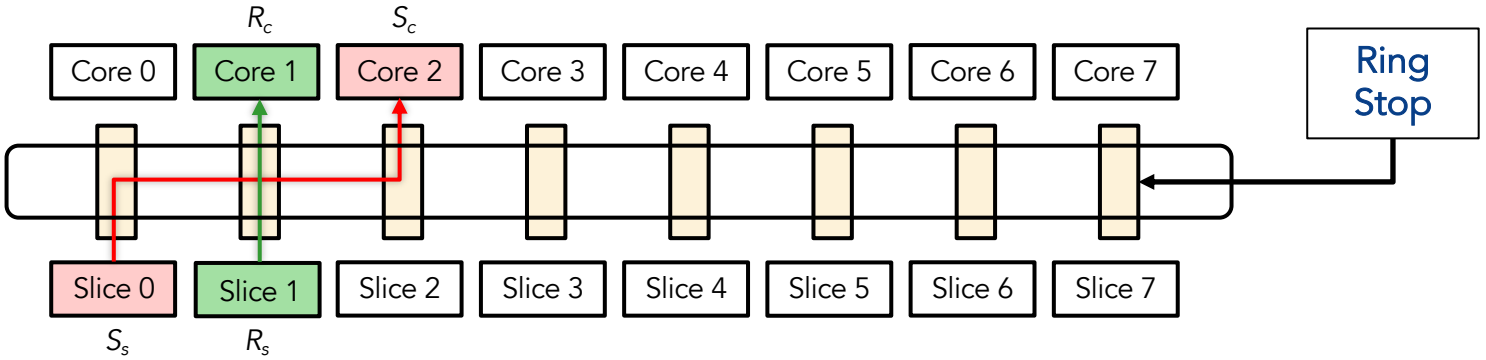
Ring Stops

- $R_c = R_s$ (and $S_s \neq R_s$) : no contention
  - Traffic between core *i* and slice *i* does not contend with cross-ring traffic.



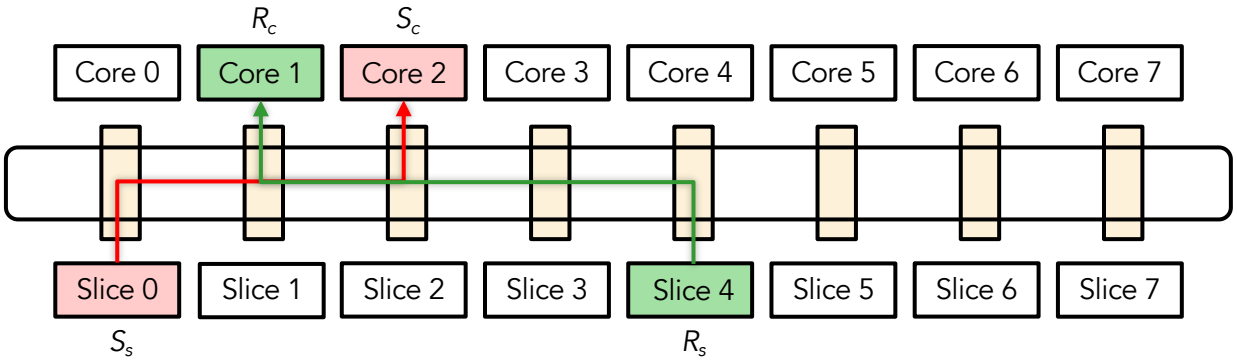
Ring Stops

- $R_c = R_s$ (and $S_s \neq R_s$) : no contention
  - Traffic between core *i* and slice *i* does not contend with cross-ring traffic.



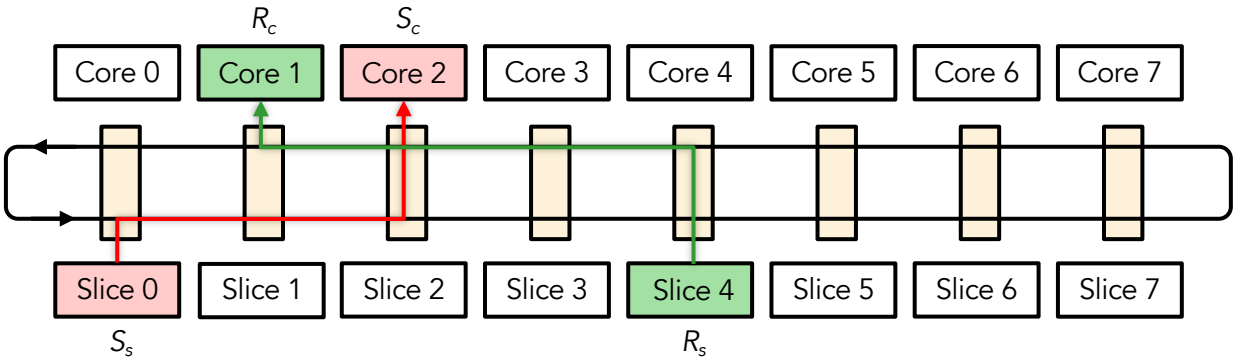
Two Directions

- Loads in opposite directions = no contention.



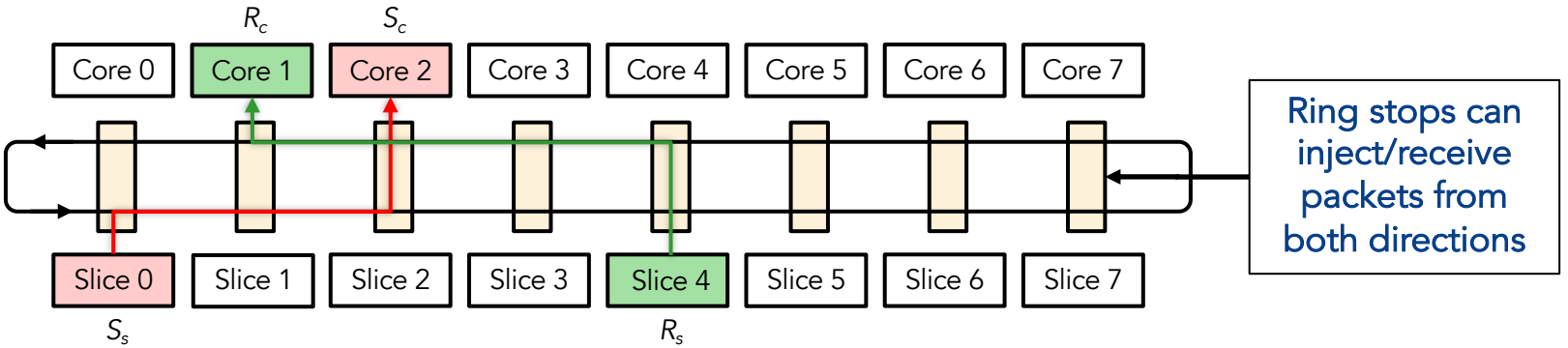
Two Directions

- Loads in opposite directions = no contention.
  - The ring has two physical flows, one for each direction.



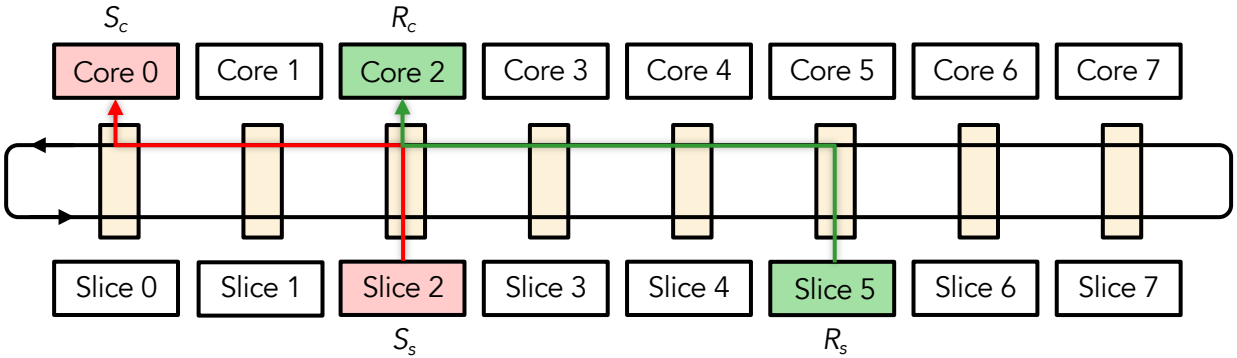
Two Directions

- Loads in opposite directions = no contention.
  - The ring has two physical flows, one for each direction.



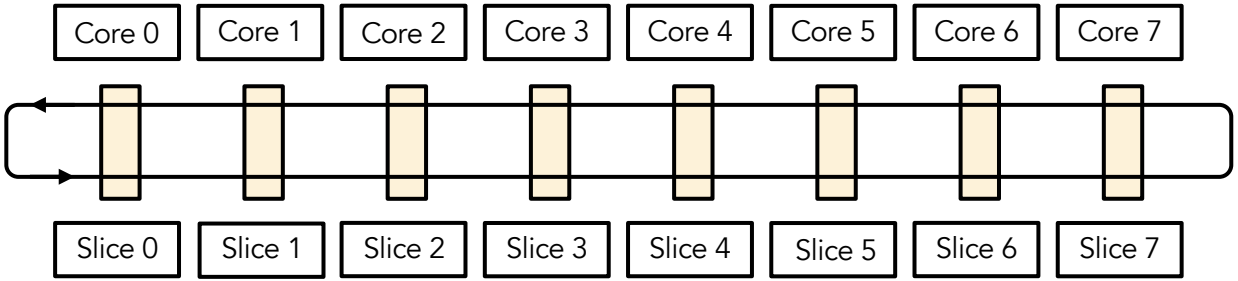
Overlapping Segments

- No overlapping = no contention.
  - Traffic on the ring only travels through the shortest path.



Checkpoint

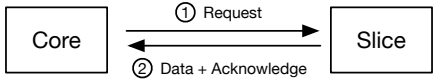
- So far: ring contention requires overlapping segments + same direction. Surprisingly, these conditions are not sufficient.



There are 4 rings

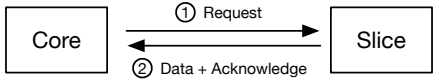
There are 4 rings

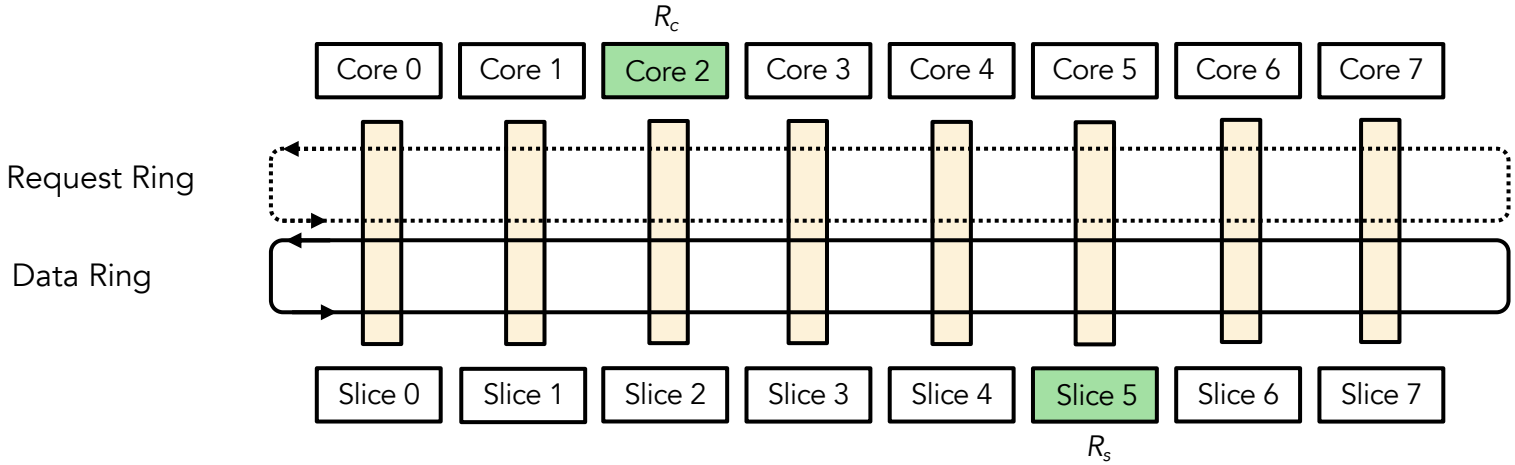
- A clean LLC load uses the request ring, the acknowledge ring and the data ring.



There are 4 rings

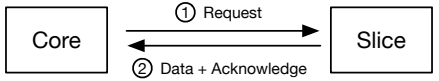
- A clean LLC load uses the request ring, the acknowledge ring and the data ring.

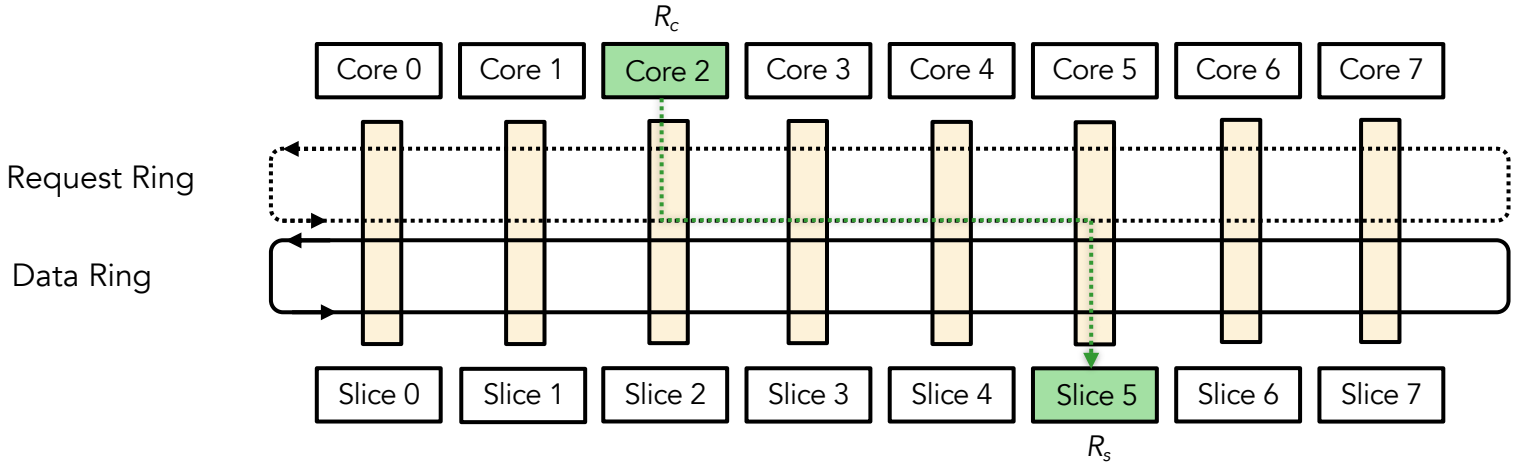




There are 4 rings

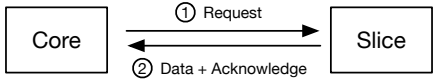
- A clean LLC load uses the request ring, the acknowledge ring and the data ring.

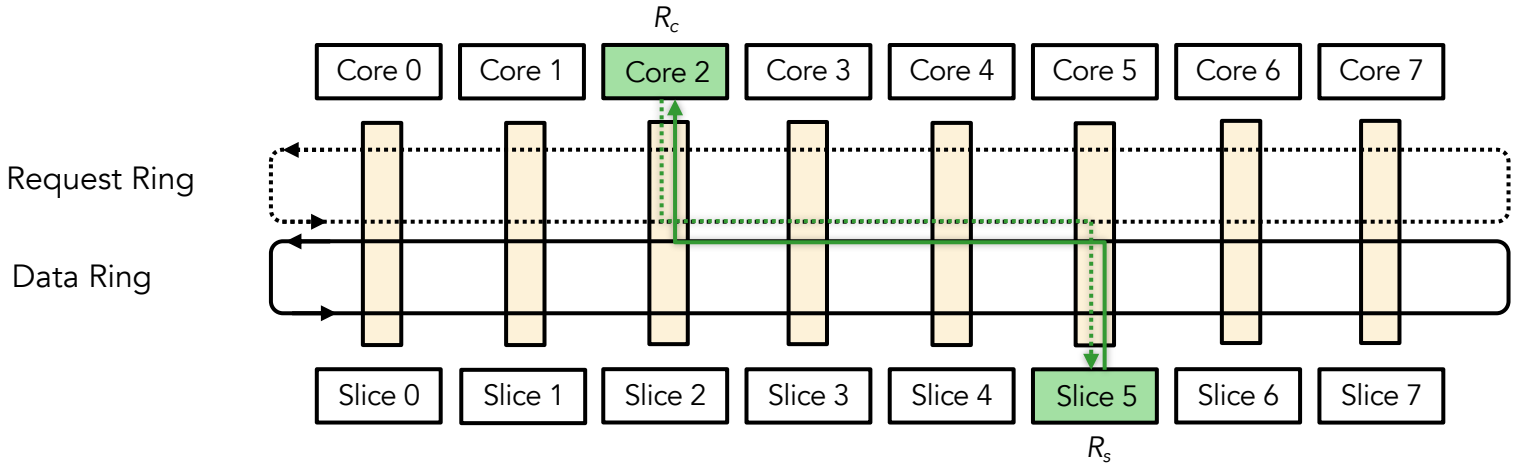




There are 4 rings

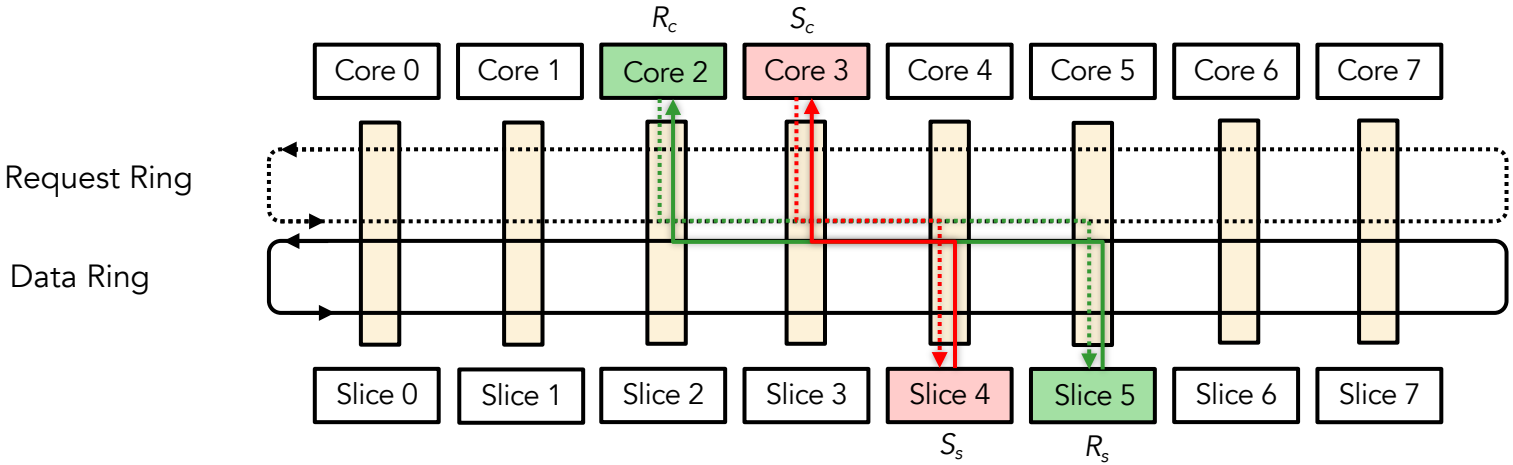
- A clean LLC load uses the request ring, the acknowledge ring and the data ring.





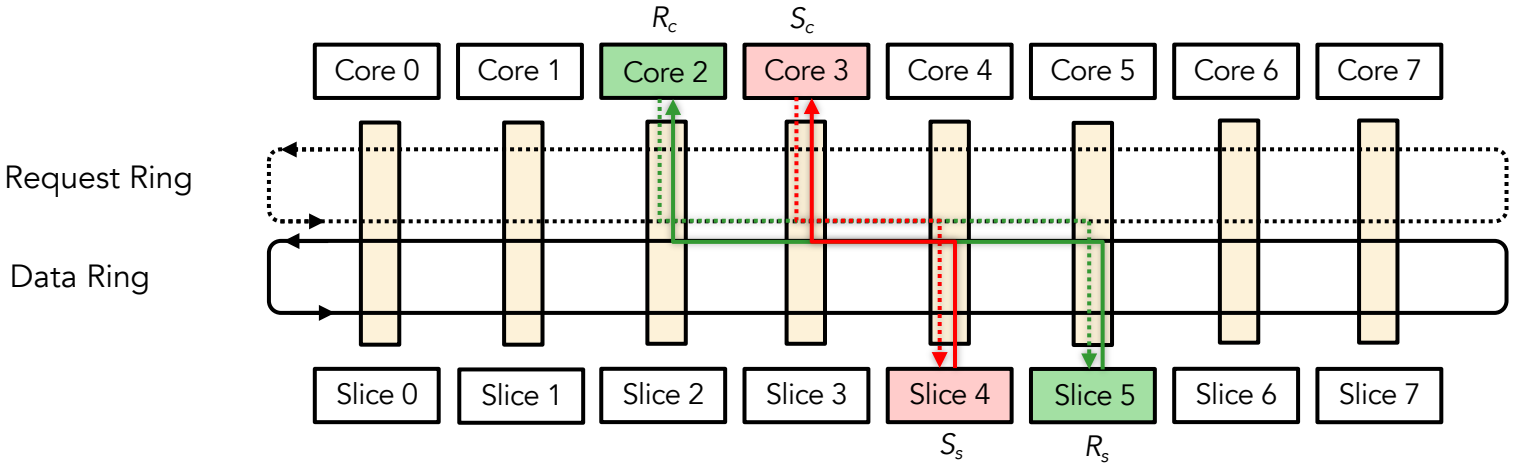
Distributed Arbitration

- Receiver envelops the sender = no contention



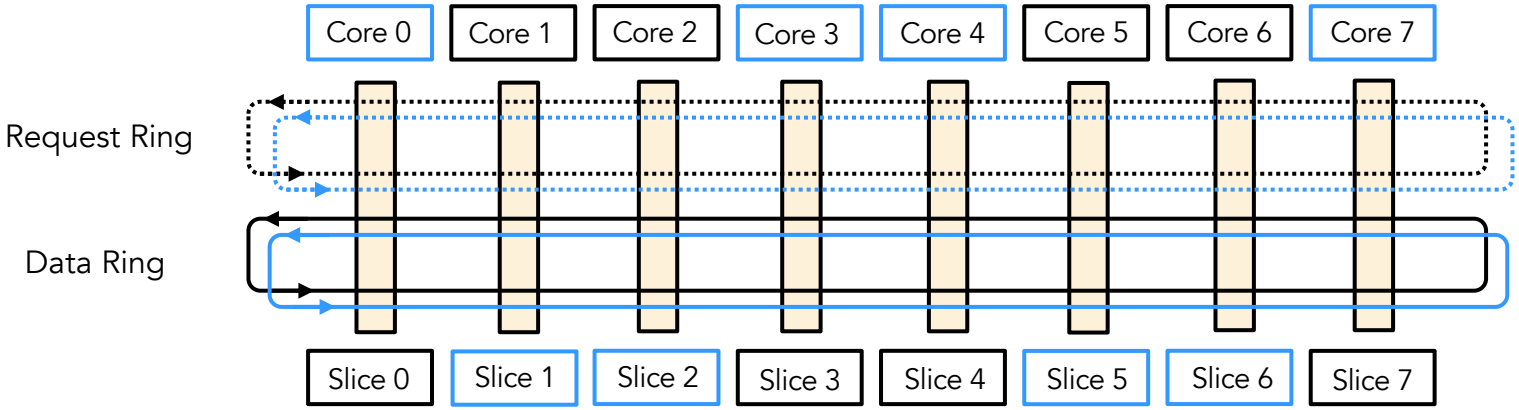
Distributed Arbitration

- Receiver envelops the sender = no contention
  - Traffic already on the ring has priority over new traffic.



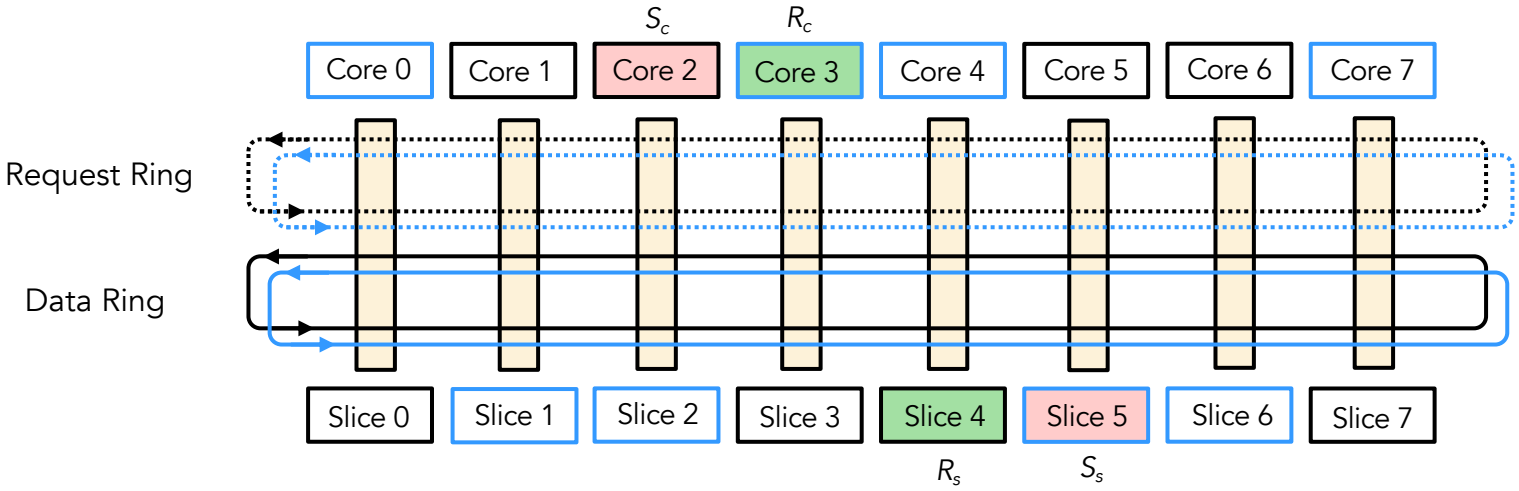
Two Lanes

- Each ring has two “lanes”, and ring stops inject traffic into different lanes depending on the cluster of its destination



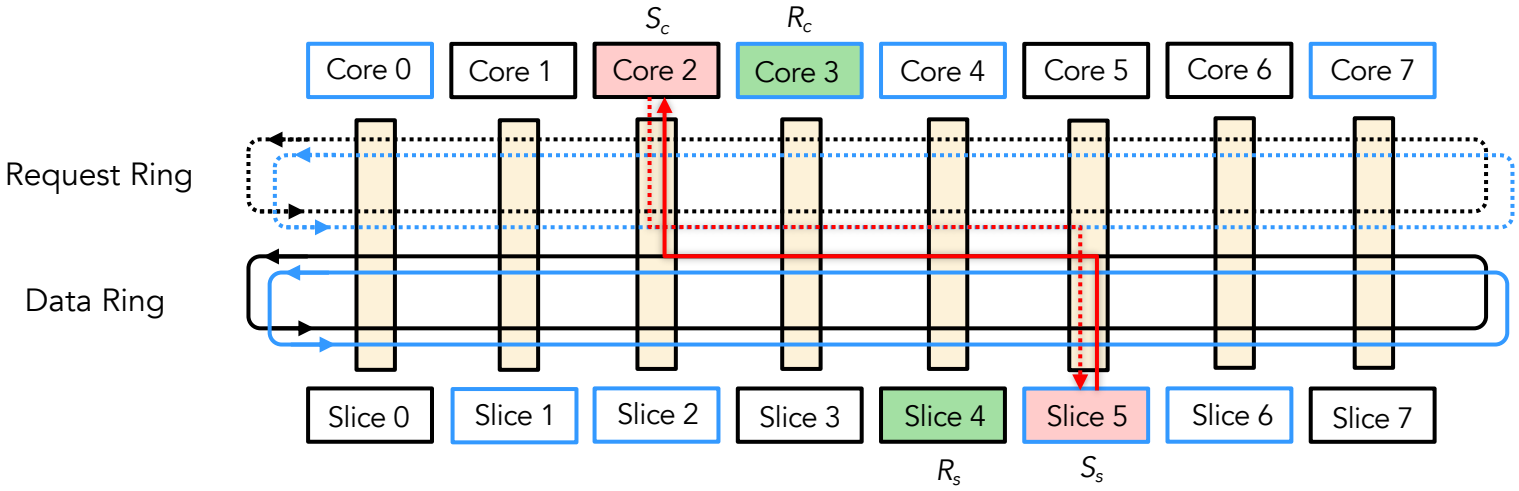
Example 1

- Contention?



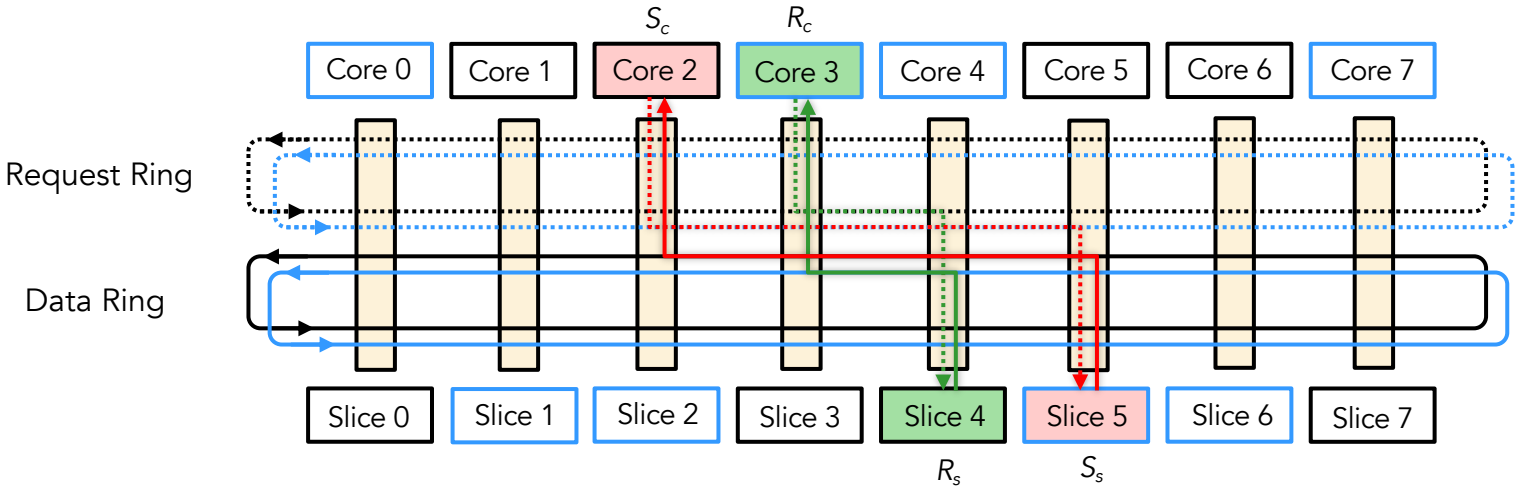
Example 1

- Contention?



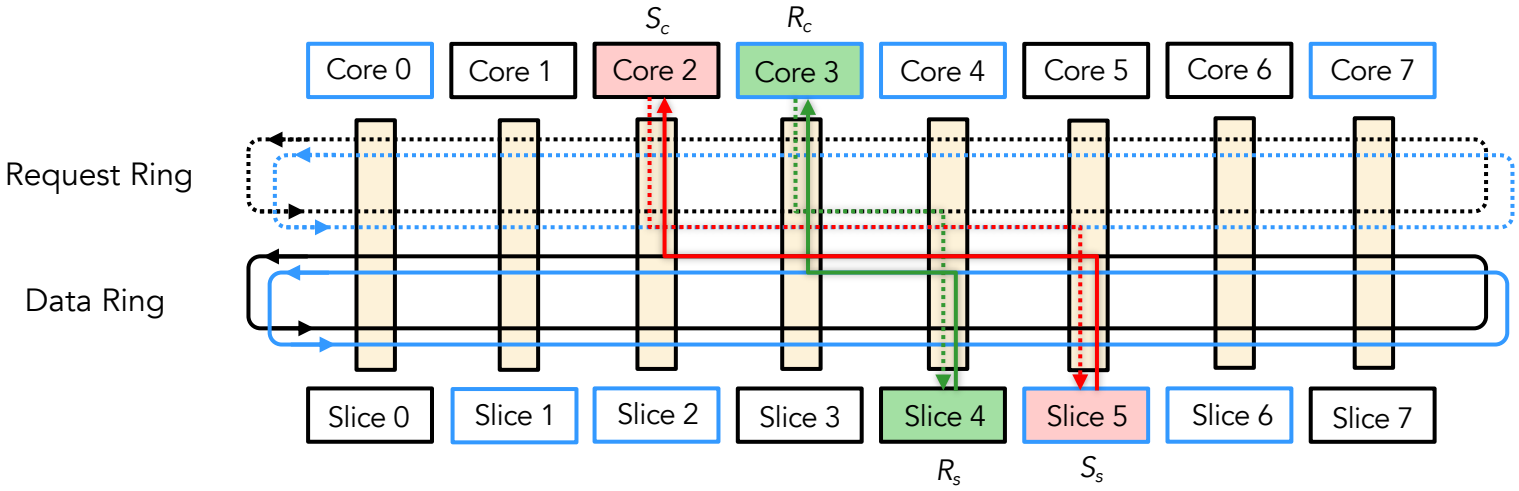
Example 1

- Contention?



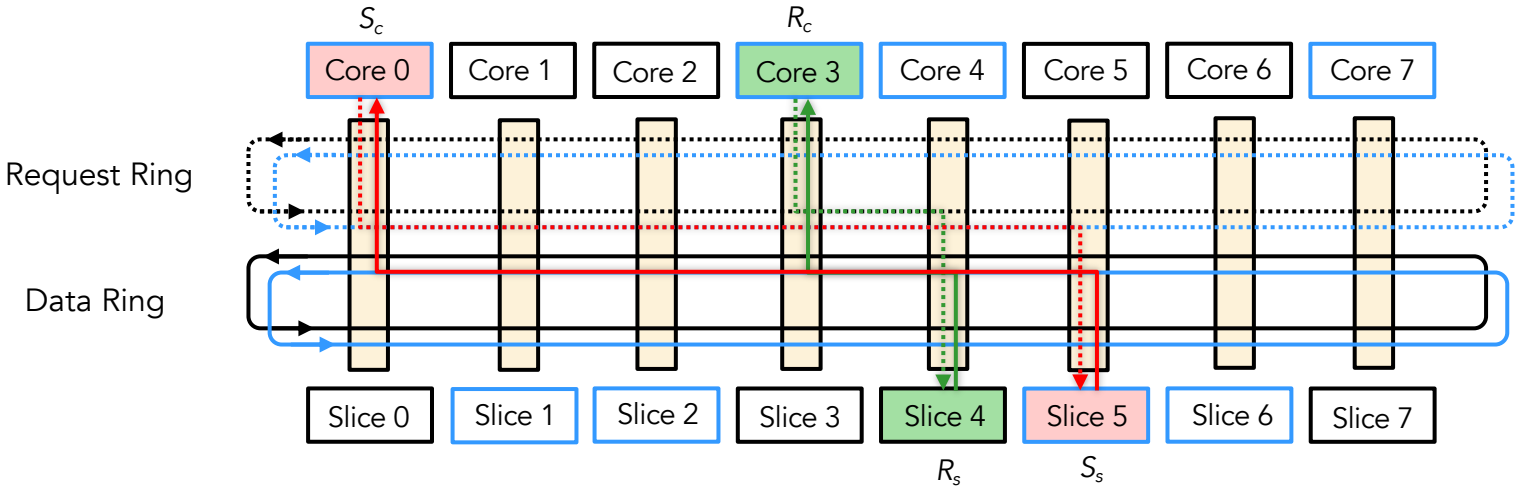
Example 1

- Contention? No, different lanes are used



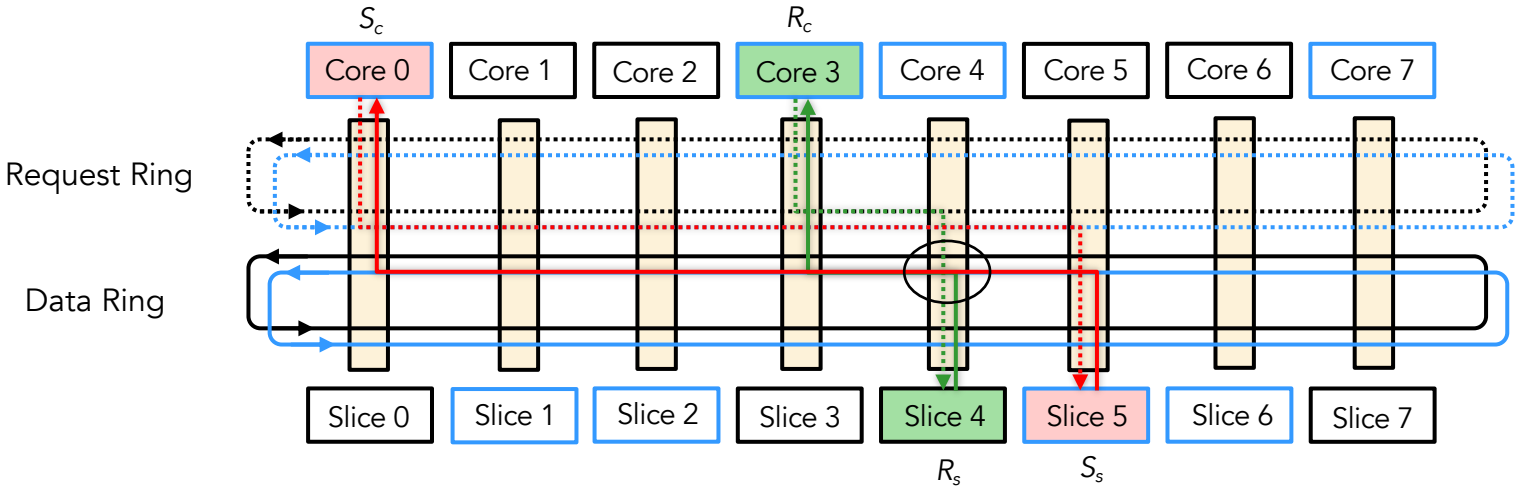
Example 2

- Contention?



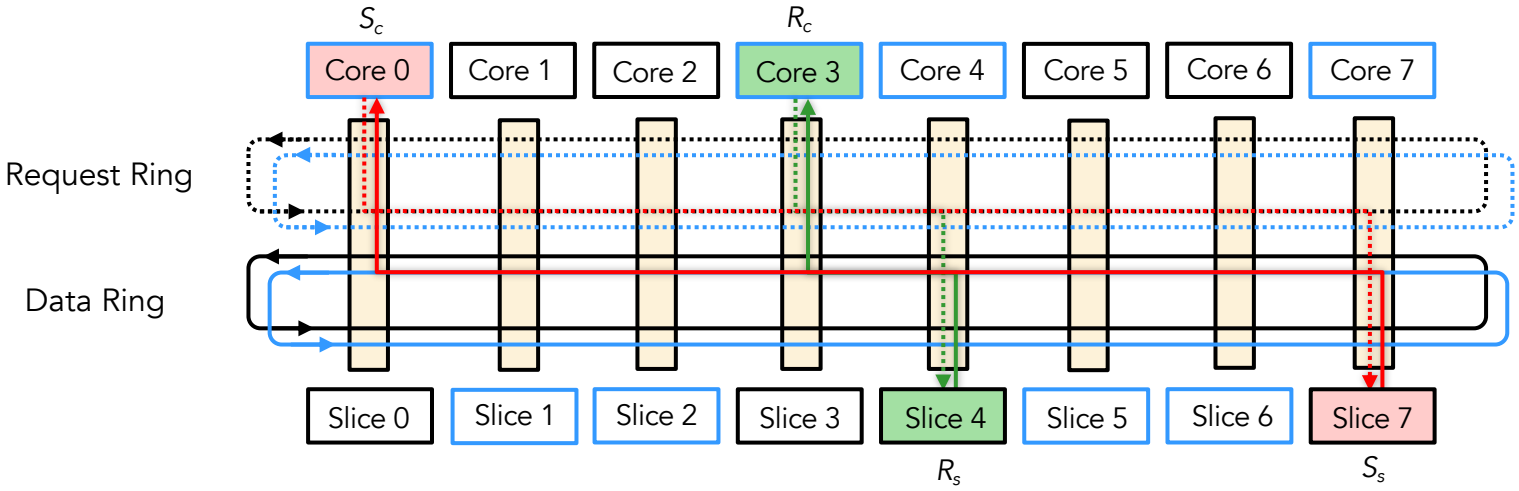
Example 2

- Contention? Yes, on the data ring!



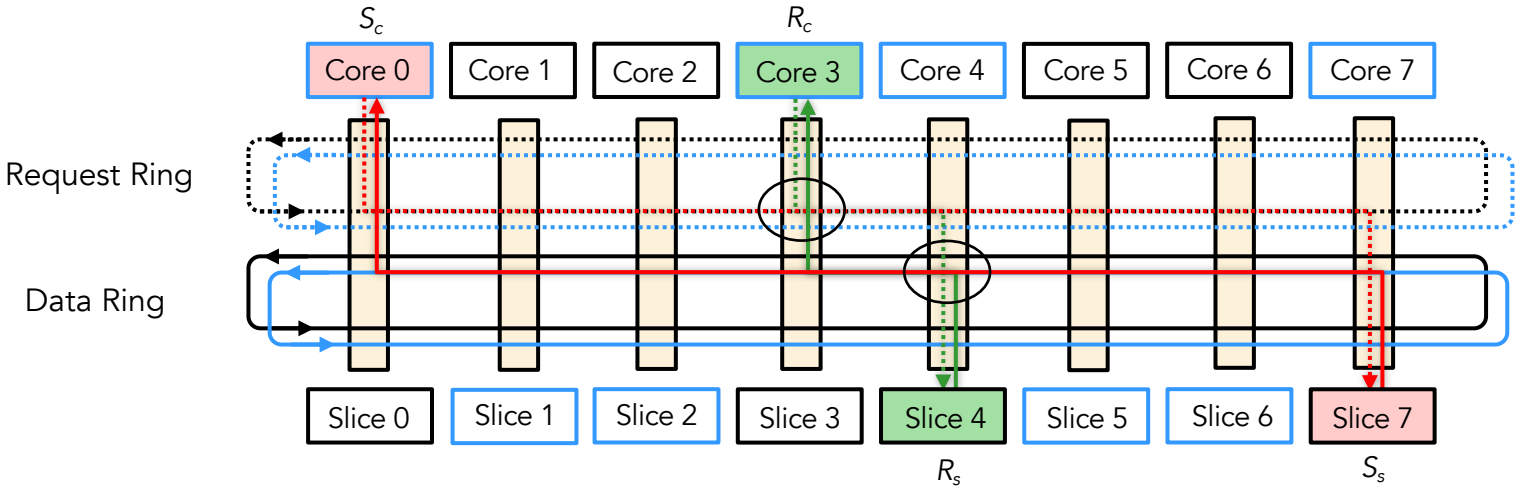
Example 3

- Contention?



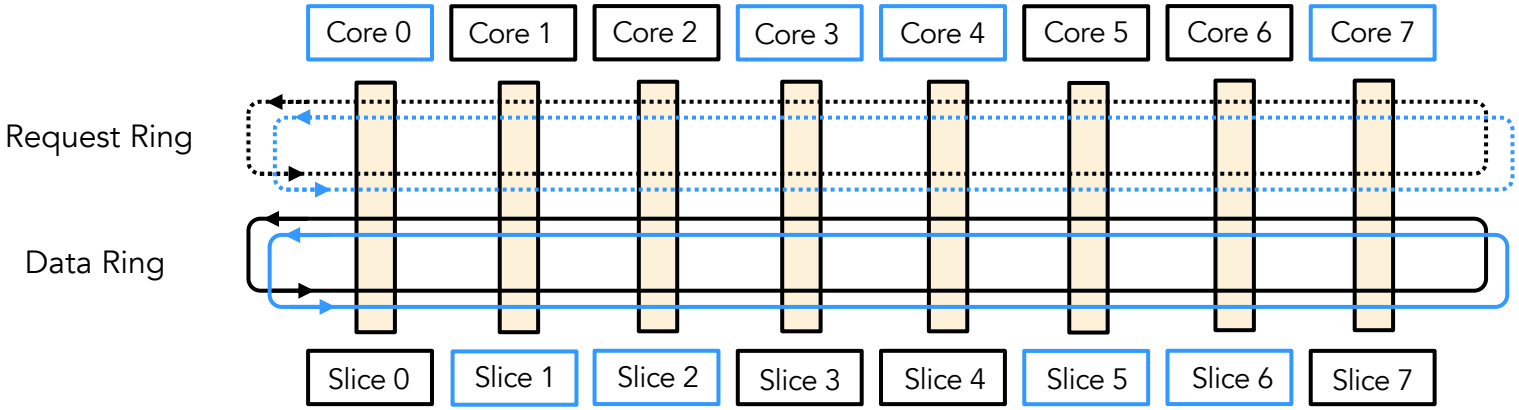
Example 3

- Contention? Yes, on the data and request rings!



Summary

- Ring contention if and only if overlapping segments + same direction + priority + same lane.

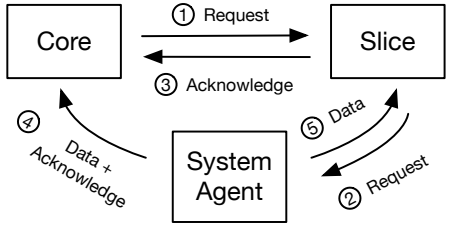


Additional observations

- Contention is larger when traffic contends on multiple rings.

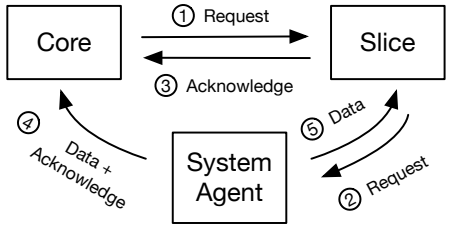
Additional observations

- Contention is larger when traffic contends on multiple rings.
- LLC misses cause additional traffic flows that create different contention patterns (see paper for details).



Additional observations

- Contention is larger when traffic contends on multiple rings.
- LLC misses cause additional traffic flows that create different contention patterns (see paper for details).
- Enabling hardware prefetchers amplifies contention.



Security Implications

- Covert channel
- Side channel attack:
  1. Side channel attack on cryptographic code
  2. Keystroke timing attack

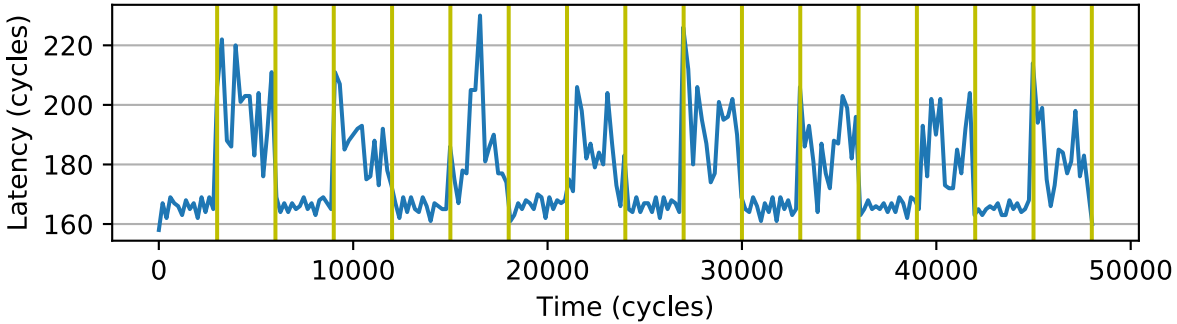
Covert Channel

Covert Channel

- Transmit a “1” → ring contention
- Transmit a “0” → idle

Covert Channel

- Transmit a “1” → ring contention
- Transmit a “0” → idle



Sender sending 010101…

- Peaks are ones
- Valleys are zeros

Covert Channel

- We tried multiple bit transmission intervals and computed the “channel capacity” metric.

Covert Channel

- We tried multiple bit transmission intervals and computed the "channel capacity" metric.
  - On our 8-core 3 GHz CPU, max capacity = 3.35 Mbps.
  - On our 4-core 4 GHz CPU, max capacity = 4.14 Mbps.

Covert Channel

- We tried multiple bit transmission intervals and computed the "channel capacity" metric.
  - On our 8-core 3 GHz CPU, max capacity = 3.35 Mbps.
  - On our 4-core 4 GHz CPU, max capacity = 4.14 Mbps.
- Largest to date for channels that do *not* rely on shared memory!

Attack on Cryptographic code

```
foreach bit b in key k do
    E1();
    if b == 1 then
        E2();
```

Attack on Cryptographic code

- If the attacker can detect E2's execution, they can leak *b*.

```
foreach bit b in key k do
    E1();
    if b == 1 then
        E2();
```

Attack on Cryptographic code

- If the attacker can detect E2's execution, they can leak *b*.
- First iteration (cold cache):

```
foreach bit b in key k do
    E1();
    if b == 1 then
        E2();
```

Attack on Cryptographic code

- If the attacker can detect E2's execution, they can leak *b*.
- First iteration (cold cache):
  - E1 loads code and data into the cache → uses the ring interconnect

```
foreach bit b in key k do
    E1();
    if b == 1 then
        E2();
```

Attack on Cryptographic code

- If the attacker can detect E2's execution, they can leak *b*.
- First iteration (cold cache):
  - E1 loads code and data into the cache → uses the ring interconnect
  - If *b=1*, E2 loads code and data into the cache → uses the ring interconnect

```
foreach bit b in key k do
    E1();
    if b == 1 then
        E2();
```

Attack on Cryptographic code

- If the attacker can detect E2's execution, they can leak *b*.
- First iteration (cold cache):
  - E1 loads code and data into the cache → uses the ring interconnect
  - If *b=1*, E2 loads code and data into the cache → uses the ring interconnect
  - If *b=0*, E2 is skipped. E1 follows and hits in the cache → does not use the ring interconnect.

```
foreach bit b in key k do
    E1();
    if b == 1 then
        E2();
```

Attack on Cryptographic code

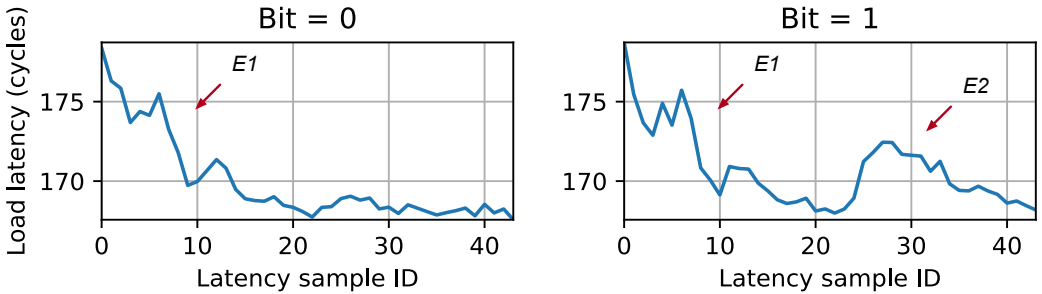
- If the attacker can detect E2's execution, they can leak *b*.
- First iteration (cold cache):
  - E1 loads code and data into the cache → uses the ring interconnect
  - If *b=1*, E2 loads code and data into the cache → uses the ring interconnect
  - If *b=0*, E2 is skipped. E1 follows and hits in the cache → does not use the ring interconnect.

```
foreach bit b in key k do
    E1();
    if b == 1 then
        E2();
```

More details in the paper

RSA square-and-multiply

- What the attacker (receiver) sees (average over 100 traces) [1]:

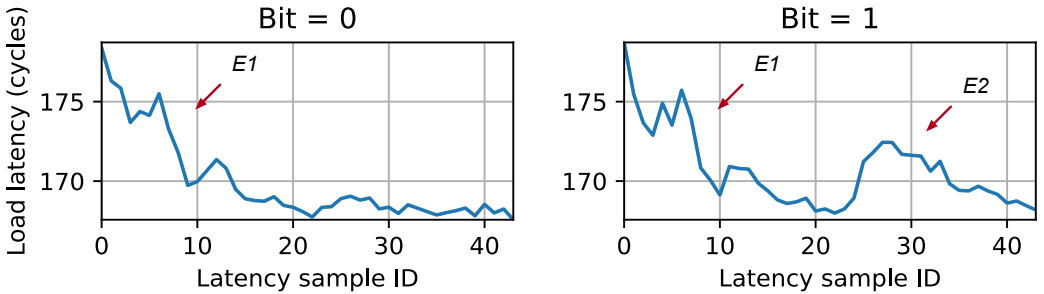



```
foreach bit b in key k do
    E1();
    if b == 1 then
        E2();
```

[1] $R_c = 2$, $R_s = 1$, $S_c = 5$

RSA square-and-multiply

- What the attacker (receiver) sees (average over 100 traces) [1]:




```
foreach bit b in key k do
    E1();
    if b == 1 then
        E2();
```

- We trained an SVM classifier to distinguish traces where bit=0 from traces where bit=1. Accuracy = 90%.

[1] $R_c = 2$, $R_s = 1$, $S_c = 5$

Keystroke timing attack

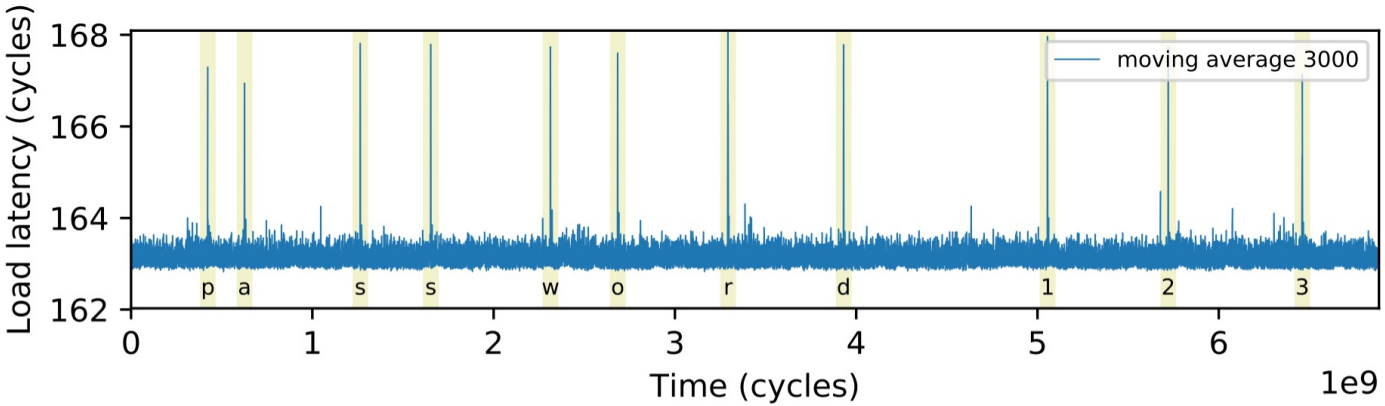
- **Goal**: detect *when* keystrokes occur to extract precise inter-keystroke timings (which can be used to leak passwords).

Keystroke timing attack

- **Goal**: detect *when* keystrokes occur to extract precise inter-keystroke timings (which can be used to leak passwords).
- Does processing a keystroke cause ring contention?

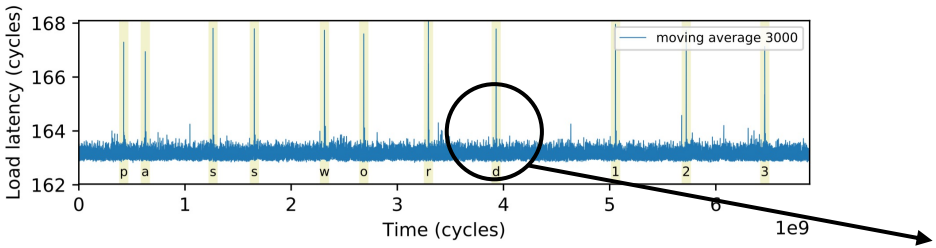
Keystroke Timing Attack

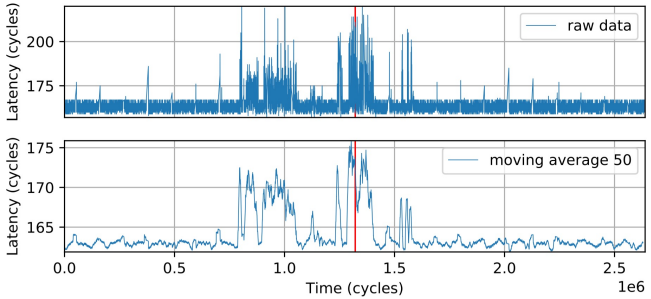
- What the attacker (receiver) sees:



Keystroke Timing Attack

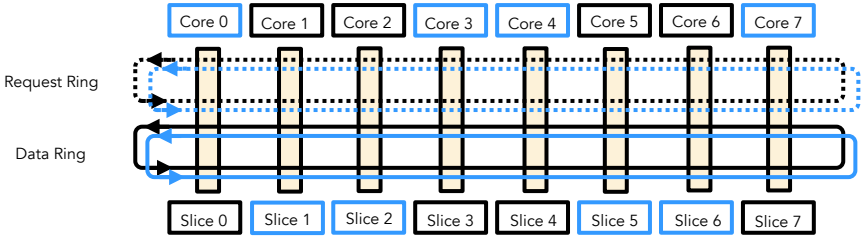
- What the attacker (receiver) sees:

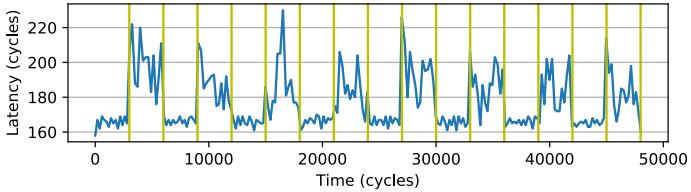


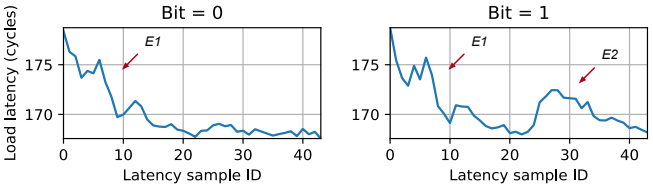


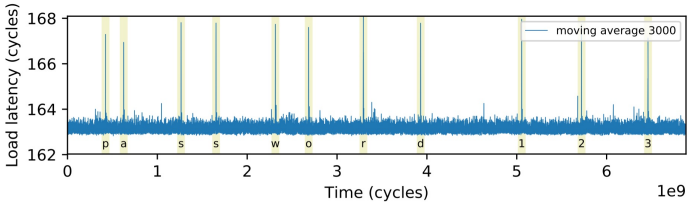
More details in the paper

Conclusion









https://github.com/FPSG-UIUC/lotr