On the Effectiveness of Same-Domain Memory Deduplication

Andreas Costi, **Brian Johannesmeyer**, Erik Bosman,
Cristiano Giuffrida, Herbert Bos

On the Effectiveness of Same-Domain Memory Deduplication

Andreas Costi, **Brian Johannesmeyer**, Erik Bosman, Cristiano Giuffrida, Herbert Bos

VU VRIJE UNIVERSITEIT AMSTERDAM

High-level takeaway

High-level takeaway

Current defenses against **memory deduplication side channel attacks** are based on **separating** trusted data from untrusted data.

High-level takeaway

Current defenses against **memory deduplication side channel attacks** are based on **separating** trusted data from untrusted data.

However, in this work, we present **two case studies** that highlight one key flaw in this defense: that it is **difficult to implement** correctly, and hence, **insufficient**.

What is memory deduplication?

What is memory deduplication?

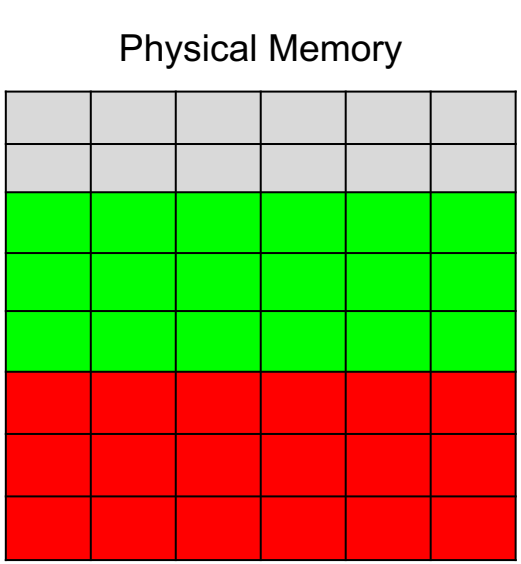
- A memory **optimization** technique used by **OSes and hypervisors**, which **scans and merges memory pages** with the **same content** across processes and virtualized guest OSes.

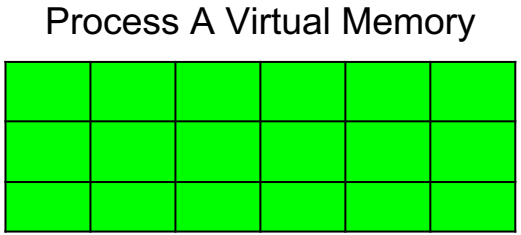
What is memory deduplication?

- A memory **optimization** technique used by **OSes and hypervisors**, which **scans and merges memory pages** with the **same content** across processes and virtualized guest OSes.
- By keeping only **one shared copy** of a page (e.g., for shared libraries, system files, etc.), it **reduces the total memory footprint** of a system.

What is memory deduplication?

- A memory **optimization** technique used by **OSes and hypervisors**, which **scans and merges memory pages** with the **same content** across processes and virtualized guest OSes.
- By keeping only **one shared copy** of a page (e.g., for shared libraries, system files, etc.), it **reduces the total memory footprint** of a system.

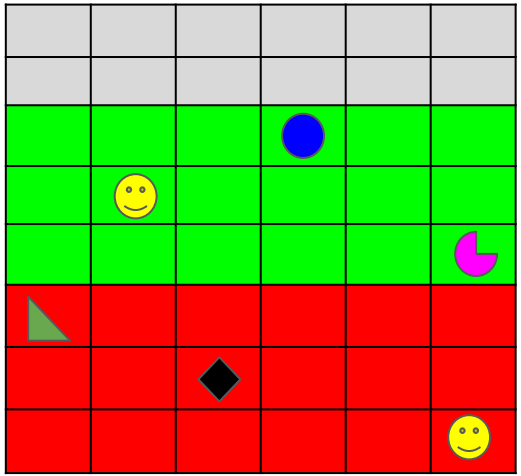




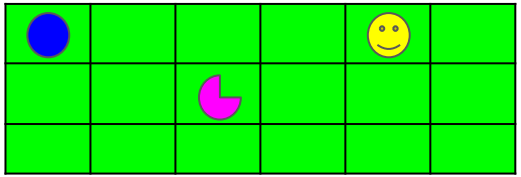
What is memory deduplication?

- A memory **optimization** technique used by **OSes and hypervisors**, which **scans and merges memory pages** with the **same content** across processes and virtualized guest OSes.
- By keeping only **one shared copy** of a page (e.g., for shared libraries, system files, etc.), it **reduces the total memory footprint** of a system.

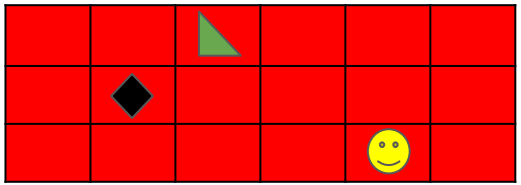
Physical Memory



Process A Virtual Memory



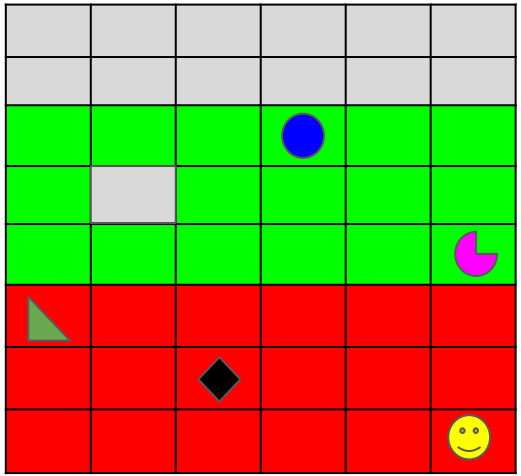
Process B Virtual Memory



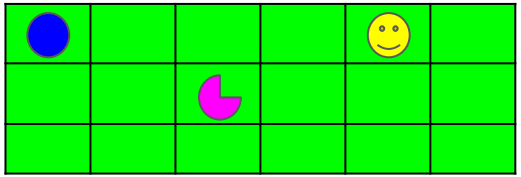
What is memory deduplication?

- A memory **optimization** technique used by **OSes and hypervisors**, which **scans and merges memory pages** with the **same content** across processes and virtualized guest OSes.
- By keeping only **one shared copy** of a page (e.g., for shared libraries, system files, etc.), it **reduces the total memory footprint** of a system.

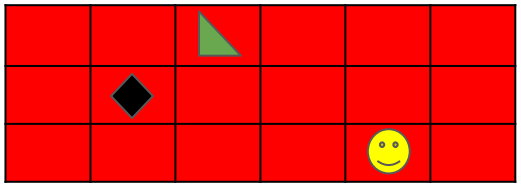
Physical Memory



Process A Virtual Memory

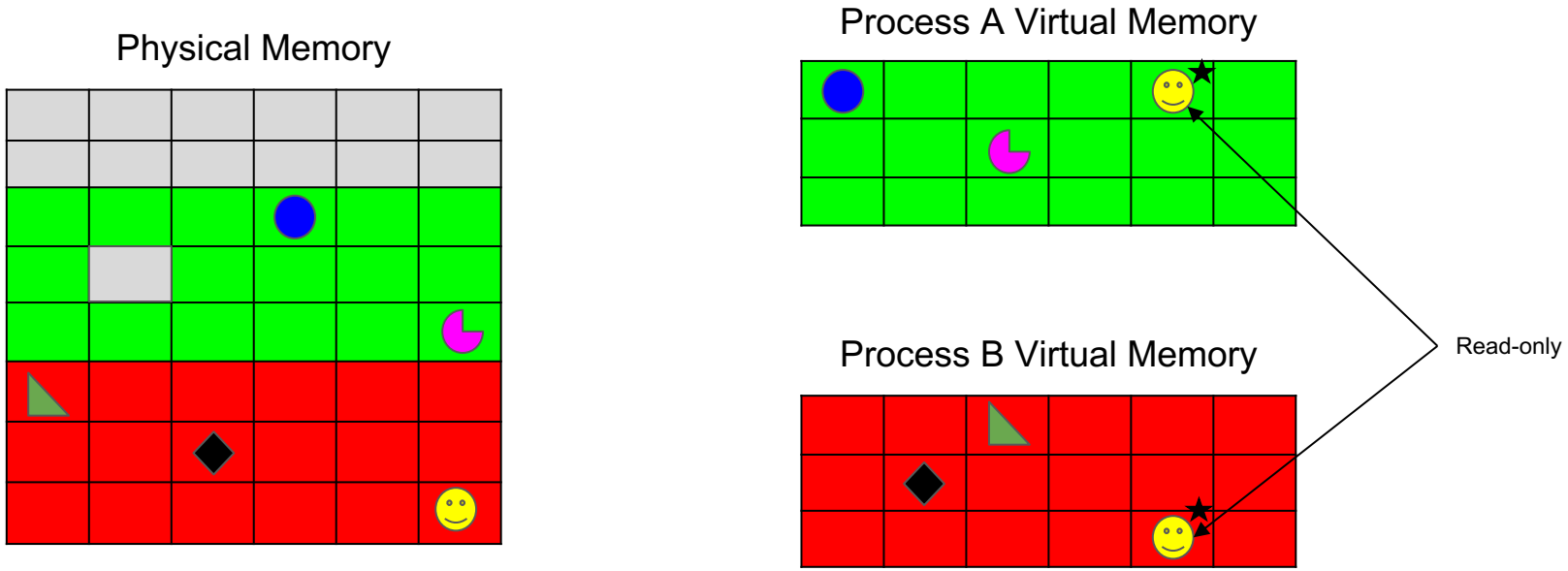


Process B Virtual Memory



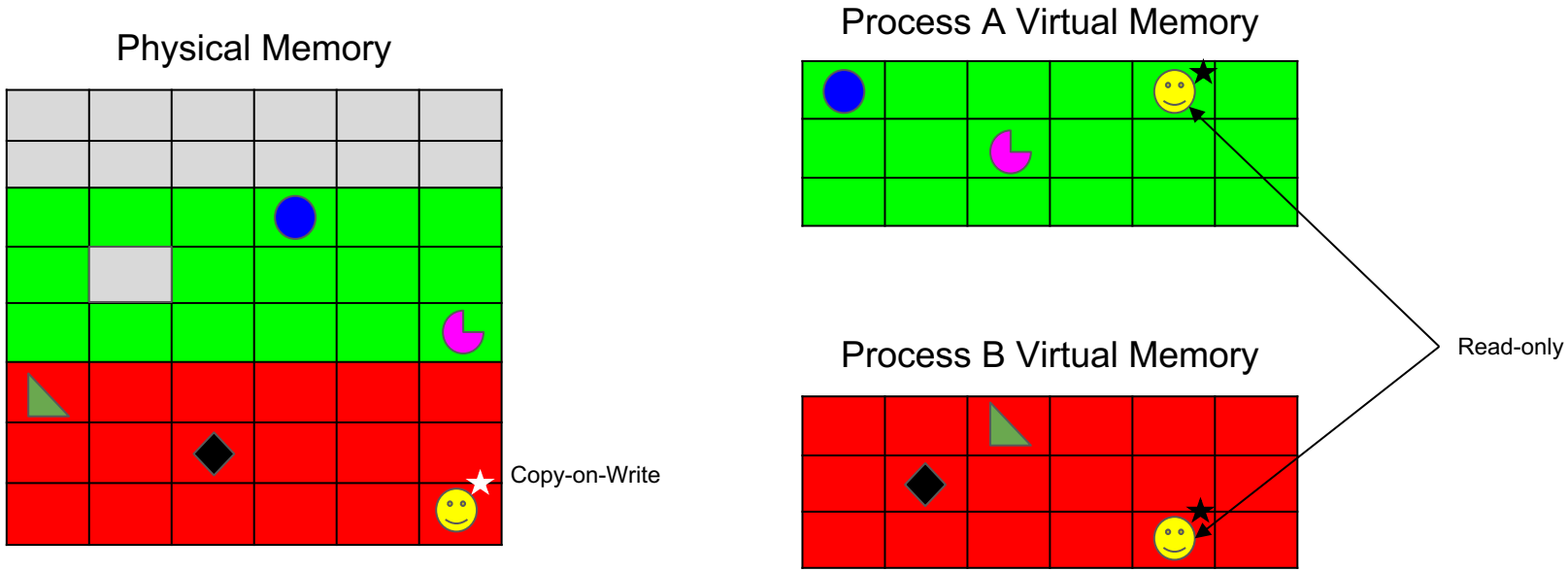
What is memory deduplication?

- A memory **optimization** technique used by **OSes and hypervisors**, which **scans and merges memory pages** with the **same content** across processes and virtualized guest OSes.
- By keeping only **one shared copy** of a page (e.g., for shared libraries, system files, etc.), it **reduces the total memory footprint** of a system.



What is memory deduplication?

- A memory **optimization** technique used by **OSes and hypervisors**, which **scans and merges memory pages** with the **same content** across processes and virtualized guest OSes.
- By keeping only **one shared copy** of a page (e.g., for shared libraries, system files, etc.), it **reduces the total memory footprint** of a system.



How is memory deduplication vulnerable?

How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

  Write to a **normal** page:

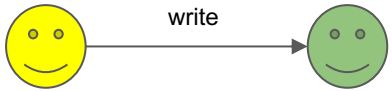
How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

Write to a **normal** page: 

How is memory deduplication vulnerable?

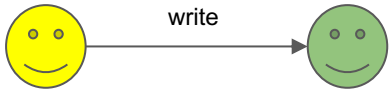
- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

Write to a **normal** page: 

How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

Write to a **normal** page:

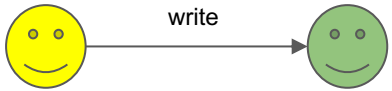


Write to a **CoW** page:

How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

Write to a **normal** page:

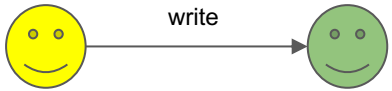


Write to a **CoW** page:

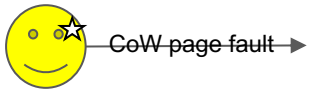
How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

Write to a **normal** page:



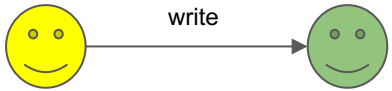
Write to a **CoW** page:



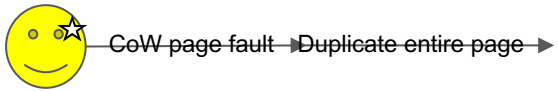
How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

Write to a **normal** page:



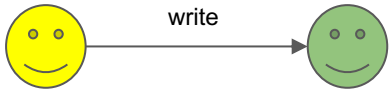
Write to a **CoW** page:



How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

Write to a **normal** page:

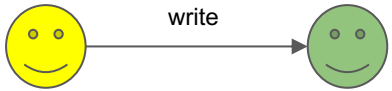


Write to a **CoW** page:

How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

Write to a **normal** page:

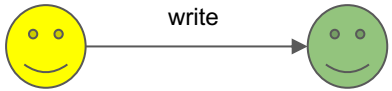


Write to a **CoW** page:

How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

Write to a **normal** page:



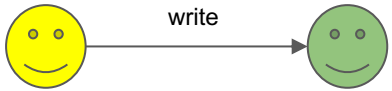
Write to a **CoW** page:

- Attackers have abused this side channel to:

How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

Write to a **normal** page:



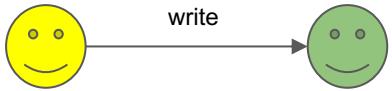
Write to a **CoW** page:

- Attackers have abused this side channel to:
  - Break ASLR,

How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

Write to a **normal** page:



Write to a **CoW** page:

- Attackers have abused this side channel to:
  - Break ASLR,
  - Break OpenSSH, GPG/APT update mechanisms, and

How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

- Attackers have abused this side channel to:
  - Break ASLR,
  - Break OpenSSH, GPG/APT update mechanisms, and
  - Escape the browser's sandbox

How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

- Attackers have abused this side channel to:
  - Break ASLR,
  - Break OpenSSH, GPG/APT update mechanisms, and
  - Escape the browser's sandbox
- In response, vendors rolled out **mitigations**, e.g.:

How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

- Attackers have abused this side channel to:
  - Break ASLR,
  - Break OpenSSH, GPG/APT update mechanisms, and
  - Escape the browser's sandbox
- In response, vendors rolled out **mitigations**, e.g.:
  - VMWare disabled **inter-VM deduplication**

How is memory deduplication vulnerable?

- However, deduplication is prone to **timing side channel attacks**, which stem from the differences between **memory write times**:

- Attackers have abused this side channel to:
  - Break ASLR,
  - Break OpenSSH, GPG/APT update mechanisms, and
  - Escape the browser's sandbox
- In response, vendors rolled out **mitigations**, e.g.:
  - VMWare disabled **inter-VM deduplication**
  - Windows disabled arbitrary **inter-process deduplication** (as we will see next)

Security domain-based deduplication

Security domain-based deduplication

- In response, **Windows only deduplicates pages** if either:

Security domain-based deduplication

- In response, **Windows only deduplicates pages** if either:
  - (1) The pages contain **safe data** (e.g., all 0s or 1s), or

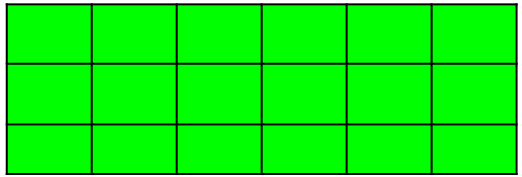
Security domain-based deduplication

- In response, **Windows only deduplicates pages** if either:
  - (1) The pages contain **safe data** (e.g., all 0s or 1s), or
  - (2) The pages are from the **same security domain** (e.g., from the same process).

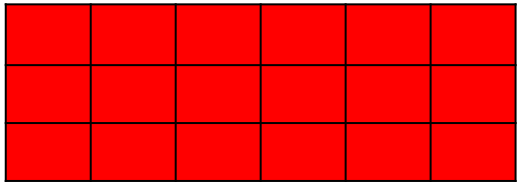
Security domain-based deduplication

- In response, **Windows only deduplicates pages** if either:
  - (1) The pages contain **safe data** (e.g., all 0s or 1s), or
  - (2) The pages are from the **same security domain** (e.g., from the same process).

Process A Virtual Memory
Security Domain: “A”



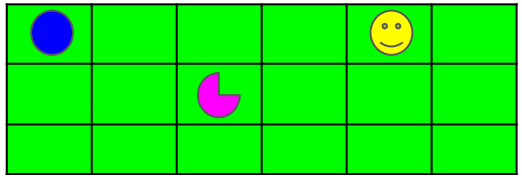
Process B Virtual Memory
Security Domain: “B”



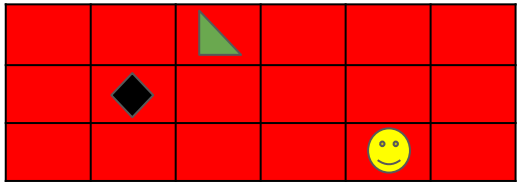
Security domain-based deduplication

- In response, **Windows only deduplicates pages** if either:
  - (1) The pages contain **safe data** (e.g., all 0s or 1s), or
  - (2) The pages are from the **same security domain** (e.g., from the same process).

Process A Virtual Memory
Security Domain: “A”



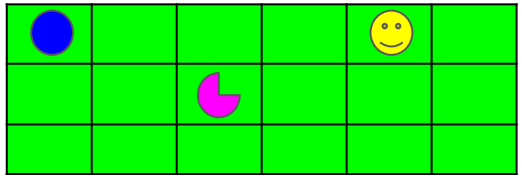
Process B Virtual Memory
Security Domain: “B”



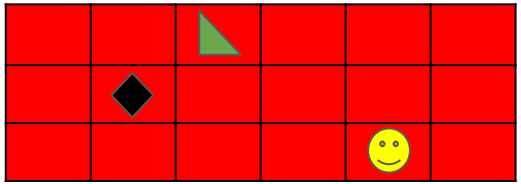
Security domain-based deduplication

- In response, **Windows only deduplicates pages** if either:
  - (1) The pages contain **safe data** (e.g., all 0s or 1s), or
  - (2) The pages are from the **same security domain** (e.g., from the same process).

Process A Virtual Memory
Security Domain: “A”



Process B Virtual Memory
Security Domain: “B”



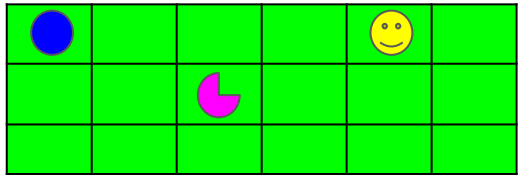
Security Domain: “A” != Security Domain: “B”

Pages not deduplicated

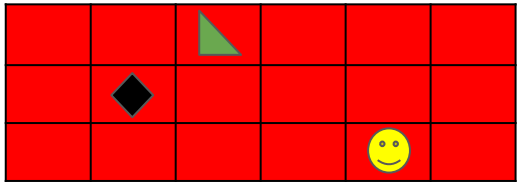
Security domain-based deduplication

- In response, **Windows only deduplicates pages** if either:
  - (1) The pages contain **safe data** (e.g., all 0s or 1s), or
  - (2) The pages are from the **same security domain** (e.g., from the same process).

Process A Virtual Memory
Security Domain: “A”



Process B Virtual Memory
Security Domain: “B”

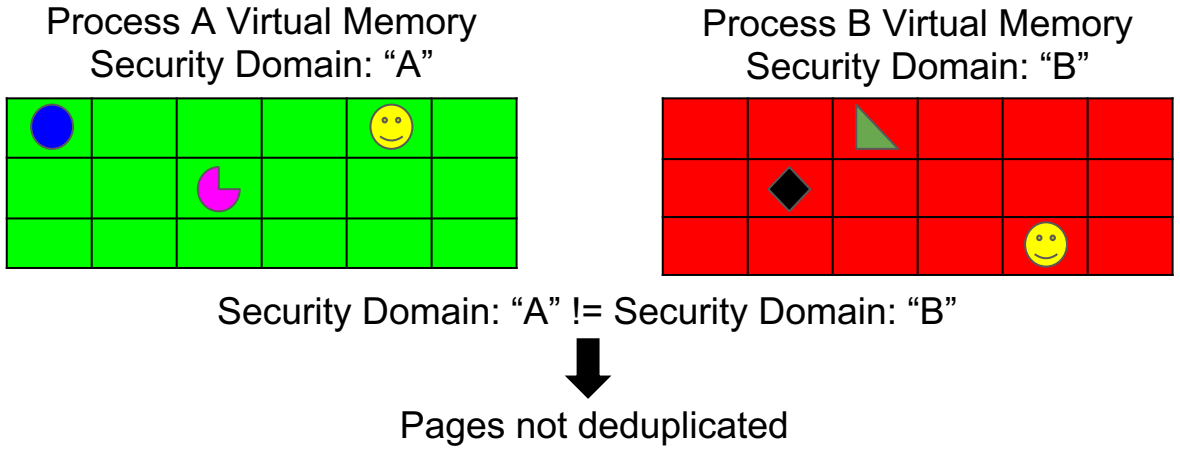


Security Domain: “A” != Security Domain: “B”

Pages not deduplicated

Security domain-based deduplication

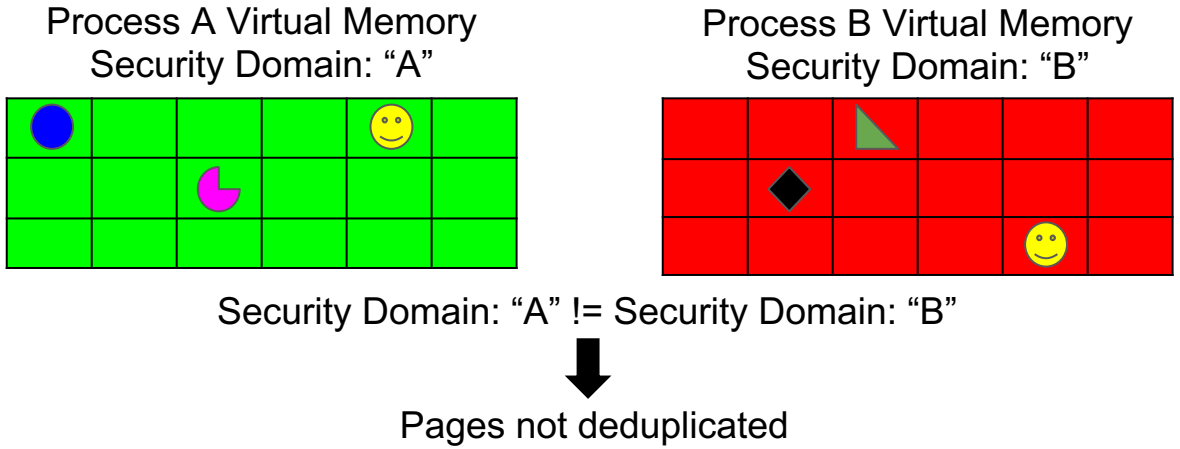
- In response, **Windows only deduplicates pages** if either:
  - (1) The pages contain **safe data** (e.g., all 0s or 1s), or
  - (2) The pages are from the **same security domain** (e.g., from the same process).



- Intra-security domain-based deduplication (e.g., same-process) is **enabled by default**.

Security domain-based deduplication

- In response, **Windows only deduplicates pages** if either:
  - (1) The pages contain **safe data** (e.g., all 0s or 1s), or
  - (2) The pages are from the **same security domain** (e.g., from the same process).

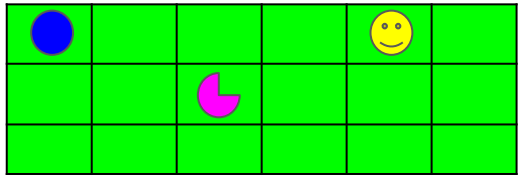


- Intra-security domain-based deduplication (e.g., same-process) is **enabled by default**.
- However, a process can **explicitly disable** it.

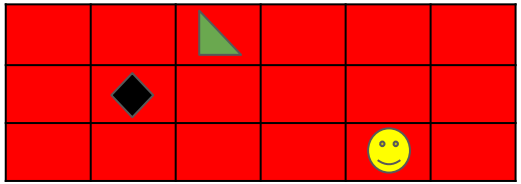
Security domain-based deduplication

- In response, **Windows only deduplicates pages** if either:
  - (1) The pages contain **safe data** (e.g., all 0s or 1s), or
  - (2) The pages are from the **same security domain** (e.g., from the same process).

Process A Virtual Memory
Security Domain: “A”



Process B Virtual Memory
Security Domain: “B”



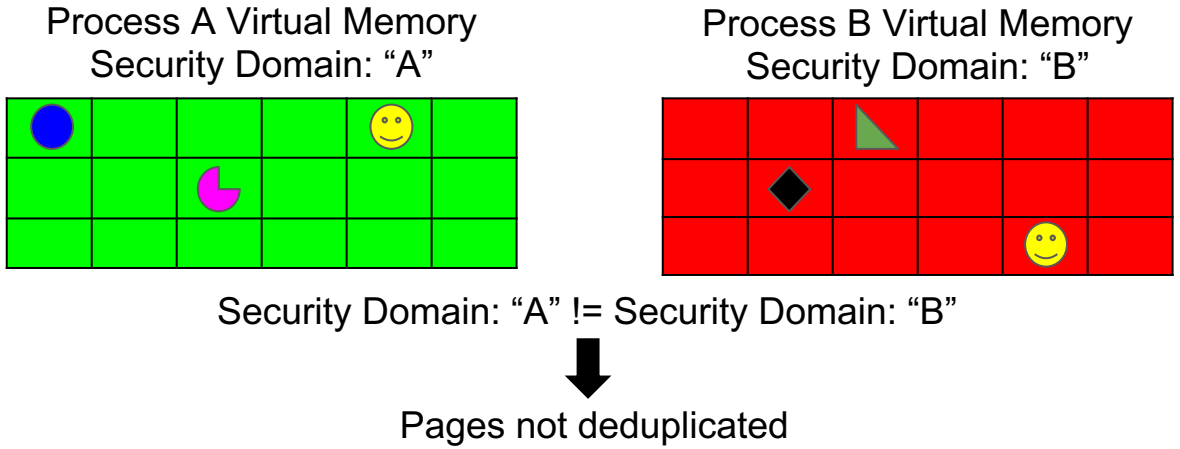
Security Domain: “A” != Security Domain: “B”

Pages not deduplicated

- Intra-security domain-based deduplication (e.g., same-process) is **enabled by default**.
- However, a process can **explicitly disable** it.
  - That way, none of a process’s own pages would deduplicate with themselves.

Security domain-based deduplication

- In response, **Windows only deduplicates pages** if either:
  - (1) The pages contain **safe data** (e.g., all 0s or 1s), or
  - (2) The pages are from the **same security domain** (e.g., from the same process).



- Intra-security domain-based deduplication (e.g., same-process) is **enabled by default**.
- However, a process can **explicitly disable** it.
  - That way, none of a process’s own pages would deduplicate with themselves.
  - This would be useful for e.g., a program handles **safe and unsafe data** within the **same process**.

Further mitigations in browsers

Further mitigations in browsers

- Browsers can piggyback on the benefits of security domain-based deduplication with their recent adoption of **site isolation**, i.e.:

Further mitigations in browsers

- Browsers can piggyback on the benefits of security domain-based deduplication with their recent adoption of **site isolation**, i.e.:
  - **Each website** open in the browser **runs in its own process**.

Further mitigations in browsers

- Browsers can piggyback on the benefits of security domain-based deduplication with their recent adoption of **site isolation**, i.e.:
  - **Each website** open in the browser **runs in its own process**.
  - As a result, **memory across different websites will not deduplicate**, eliminating the deduplication attack vector.

Further mitigations in browsers

- Browsers can piggyback on the benefits of security domain-based deduplication with their recent adoption of **site isolation**, i.e.:
  - **Each website** open in the browser **runs in its own process**.
  - As a result, **memory across different websites will not deduplicate**, eliminating the deduplication attack vector.
  - However, **adoption** of strict site isolation has been **slow**.

Further mitigations in browsers

- Browsers can piggyback on the benefits of security domain-based deduplication with their recent adoption of **site isolation**, i.e.:
  - **Each website** open in the browser **runs in its own process**.
  - As a result, **memory across different websites will not deduplicate**, eliminating the deduplication attack vector.
  - However, **adoption** of strict site isolation has been **slow**.
    - In particular, at the time of our research, **Firefox had a partial implementation** of site isolation.

Further mitigations in browsers

- Browsers can piggyback on the benefits of security domain-based deduplication with their recent adoption of **site isolation**, i.e.:
  - **Each website** open in the browser **runs in its own process**.
  - As a result, **memory across different websites will not deduplicate**, eliminating the deduplication attack vector.
  - However, **adoption** of strict site isolation has been **slow**.
    - In particular, at the time of our research, **Firefox had a partial implementation** of site isolation.
    - However, this was **fixed** in **November 2021** (Firefox v94.0).

Further mitigations in browsers

- Browsers can piggyback on the benefits of security domain-based deduplication with their recent adoption of **site isolation**, i.e.:
  - **Each website** open in the browser **runs in its own process**.
  - As a result, **memory across different websites will not deduplicate**, eliminating the deduplication attack vector.
  - However, **adoption** of strict site isolation has been **slow**.
    - In particular, at the time of our research, **Firefox had a partial implementation** of site isolation.
    - However, this was **fixed** in **November 2021** (Firefox v94.0).
- Moreover, to thwart side channel attacks, browsers **throttle the granularity of native timers**.

Further mitigations in browsers

- Browsers can piggyback on the benefits of security domain-based deduplication with their recent adoption of **site isolation**, i.e.:
  - **Each website** open in the browser **runs in its own process**.
  - As a result, **memory across different websites will not deduplicate**, eliminating the deduplication attack vector.
  - However, **adoption** of strict site isolation has been **slow**.
    - In particular, at the time of our research, **Firefox had a partial implementation** of site isolation.
    - However, this was **fixed** in **November 2021** (Firefox v94.0).
- Moreover, to thwart side channel attacks, browsers **throttle the granularity of native timers**.
  - As a result, an attacker **cannot accurately time write operations** via e.g., performance.now().

Further mitigations in browsers

- Browsers can piggyback on the benefits of security domain-based deduplication with their recent adoption of **site isolation**, i.e.:
  - **Each website** open in the browser **runs in its own process**.
  - As a result, **memory across different websites will not deduplicate**, eliminating the deduplication attack vector.
  - However, **adoption** of strict site isolation has been **slow**.
    - In particular, at the time of our research, **Firefox had a partial implementation** of site isolation.
    - However, this was **fixed** in **November 2021** (Firefox v94.0).
- Moreover, to thwart side channel attacks, browsers **throttle the granularity of native timers**.
  - As a result, an attacker **cannot accurately time write operations** via e.g., performance.now().
  - However, later work bypasses this by finding **alternative** sources of fantastic **timers**.

Further mitigations in browsers

- Browsers can piggyback on the benefits of security domain-based deduplication with their recent adoption of **site isolation**, i.e.:
  - **Each website** open in the browser **runs in its own process**.
  - As a result, **memory across different websites will not deduplicate**, eliminating the deduplication attack vector.
  - However, **adoption** of strict site isolation has been **slow**.
    - In particular, at the time of our research, **Firefox had a partial implementation** of site isolation.
    - However, this was **fixed** in **November 2021** (Firefox v94.0).
- Moreover, to thwart side channel attacks, browsers **throttle the granularity of native timers**.
  - As a result, an attacker **cannot accurately time write operations** via e.g., performance.now().
  - However, later work bypasses this by finding **alternative** sources of fantastic **timers**.
  - E.g., by using a **SharedArrayBuffer counter**, one thread can increment a “timer” value in a loop, while the other thread polls the “timer”.

Overview

Overview

- This work asks the question: are these mitigations **sufficient**?

Overview

- This work asks the question: are these mitigations **sufficient**?
- The answer: it depends on your **threat model**!

Overview

- This work asks the question: are these mitigations **sufficient**?
- The answer: it depends on your **threat model**!
- If you assume that all processes will **never intermingle trusted and untrusted data** — then yes, it is sufficient!

Overview

- This work asks the question: are these mitigations **sufficient**?
- The answer: it depends on your **threat model**!
- If you assume that all processes will **never intermingle trusted and untrusted data** — then yes, it is sufficient!
- However, we present **two case studies** that demonstrate that this assumption **does not hold in practice**.

Overview

- This work asks the question: are these mitigations **sufficient**?
- The answer: it depends on your **threat model**!
- If you assume that all processes will **never intermingle trusted and untrusted data** — then yes, it is sufficient!
- However, we present **two case studies** that demonstrate that this assumption **does not hold in practice**.

Client-server scenario: Overview

Client-server scenario: Overview

- In this scenario, the **server is the victim** and the **client is the attacker**.

Client-server scenario: Overview

- In this scenario, the **server is the victim** and the **client is the attacker**.
- Setup:

Client-server scenario: Overview

- In this scenario, the **server is the victim** and the **client is the attacker**.
- Setup:
  - The **server**: stores **untrusted data** from the client **alongside its own secret data**.

Client-server scenario: Overview

- In this scenario, the **server is the victim** and the **client is the attacker**.
- Setup:
  - The **server**: stores **untrusted data** from the client **alongside its own secret data**.
  - The **client**: (1) **sends data** to the server, and (2) **times** the server's responses.

Client-server scenario: Overview

- In this scenario, the **server is the victim** and the **client is the attacker**.
- Setup:
  - The **server**: stores **untrusted data** from the client **alongside its own secret data**.
  - The **client**: (1) **sends data** to the server, and (2) **times** the server's responses.
- This resembles e.g., a server such as a nginx running a key-value store, which untrusted clients can connect to.

Client-server scenario: Exploit steps

Client-server scenario: Exploit steps

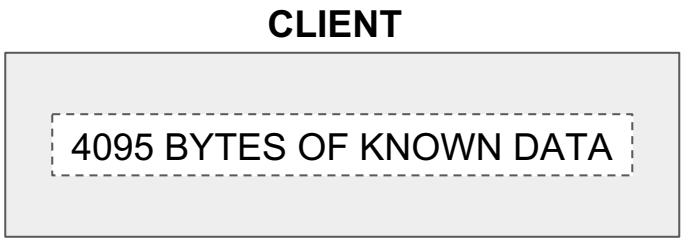
1. **Prime**: The client sends data to the server, including:

Client-server scenario: Exploit steps

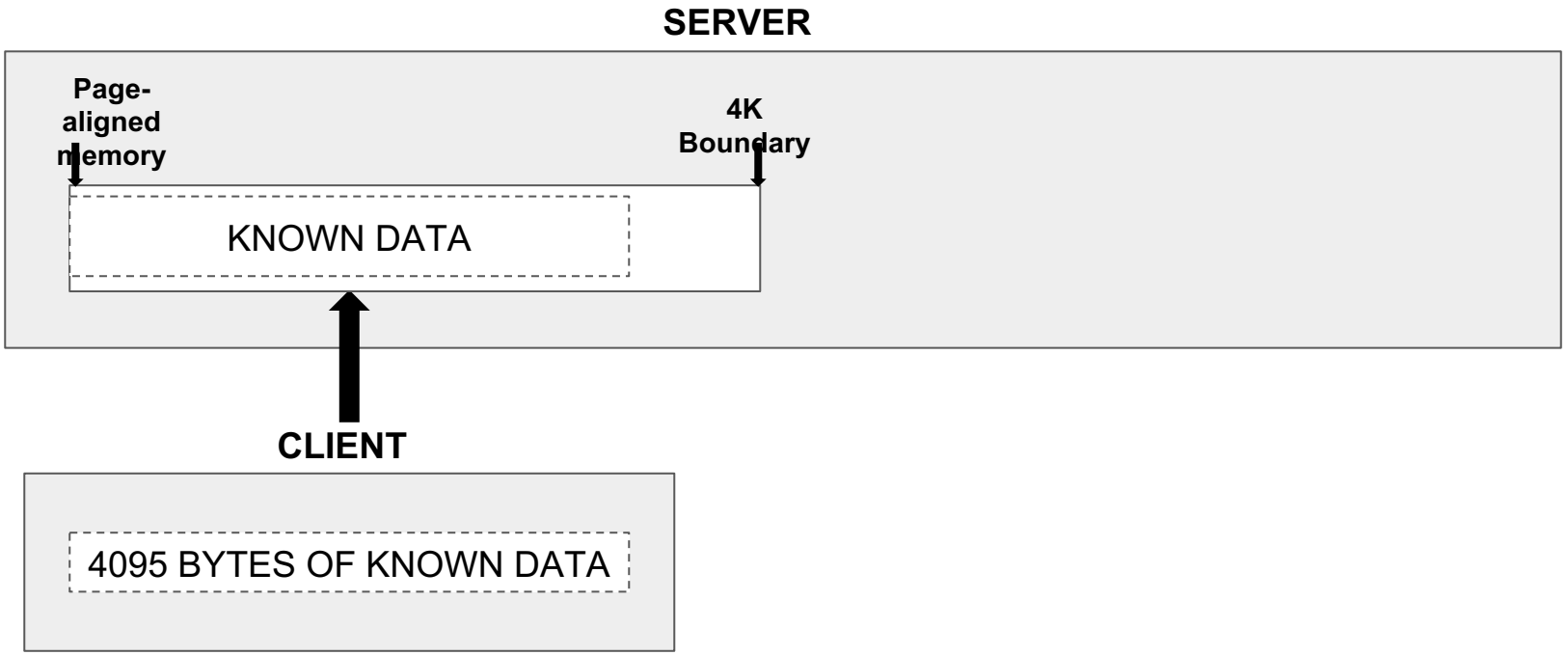
1. **Prime**: The client sends data to the server, including:
    a. Data that will be stored next to secret data, i.e., on the **secret page**

Client-server scenario: Creating the secret page

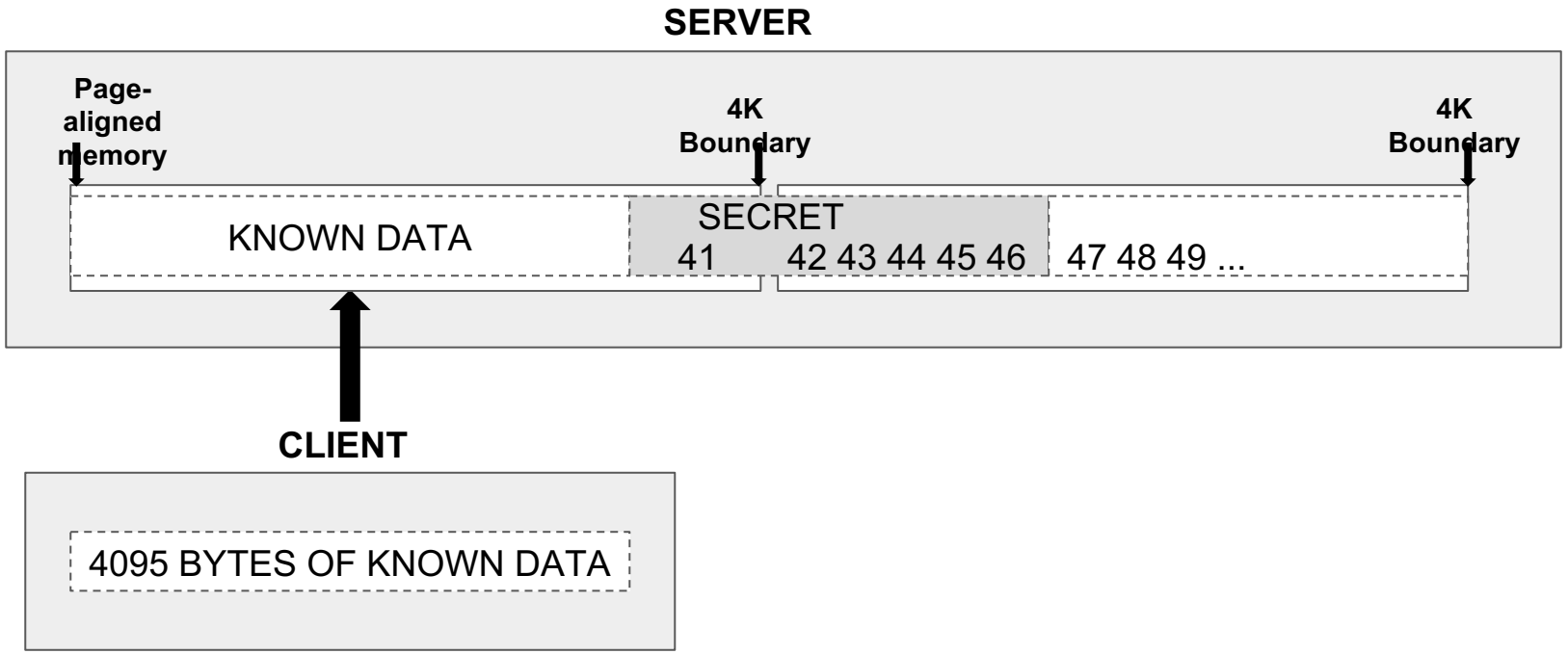
Client-server scenario: Creating the secret page



Client-server scenario: Creating the secret page



Client-server scenario: Creating the secret page



Client-server scenario: Exploit steps

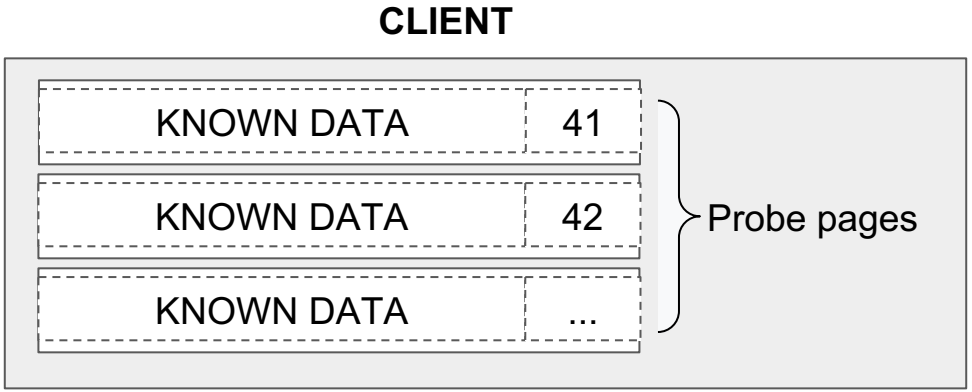
1. **Prime**: The client sends data to the server, including:
    a. Data that will be stored next to secret data, i.e., on the **secret page**

Client-server scenario: Exploit steps

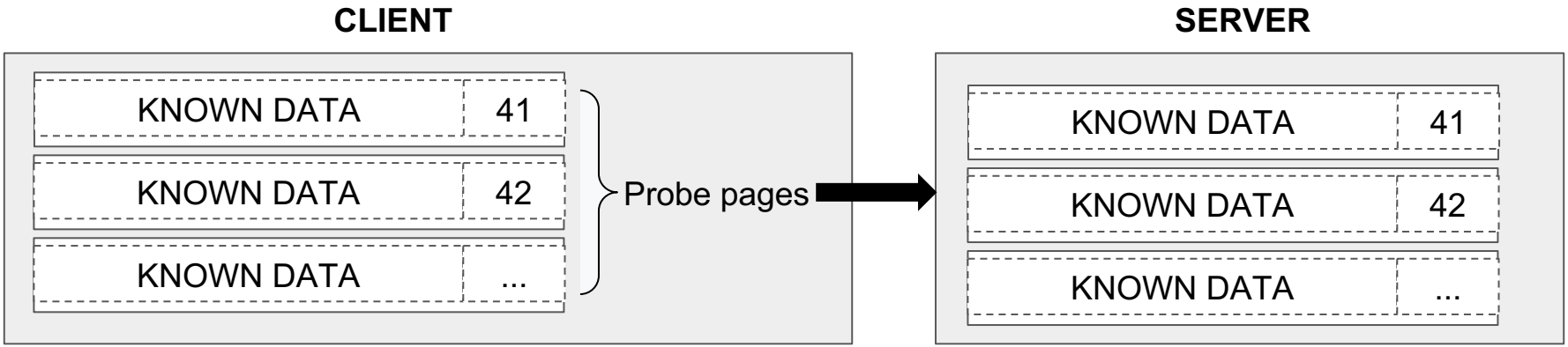
1. **Prime**: The client sends data to the server, including:
    a. Data that will be stored next to secret data, i.e., on the **secret page**
    b. Multiple **probe pages**

Client-server scenario: Creating probe pages

Client-server scenario: Creating probe pages



Client-server scenario: Creating probe pages



Client-server scenario: Exploit steps

1. **Prime**: The client sends data to the server, including:
    a. Data that will be stored next to secret data, i.e., on the **secret page**
    b. Multiple **probe pages**

Client-server scenario: Exploit steps

1. **Prime**: The client sends data to the server, including:
    a. Data that will be stored next to secret data, i.e., on the **secret page**
    b. Multiple **probe pages**
2. The client **waits for a deduplication pass** to occur.

Client-server scenario: Exploit steps

1. **Prime**: The client sends data to the server, including:
   a. Data that will be stored next to secret data, i.e., on the **secret page**
   b. Multiple **probe pages**
2. The client **waits for a deduplication pass** to occur.
3. **Probe**: The client updates its probe pages and measures **how long** it takes for the **server to respond**.

Client-server scenario: Exploit steps

1. **Prime**: The client sends data to the server, including:
   a. Data that will be stored next to secret data, i.e., on the **secret page**
   b. Multiple **probe pages**
2. The client **waits for a deduplication pass** to occur.
3. **Probe**: The client updates its probe pages and measures **how long** it takes for the **server to respond**.
   a. If it was **slow**, then the data was **deduplicated**, so: **secret data = probe data.**

Client-server scenario: Probing

1. **Prime**: The client sends data to the server, including:
   a. Data that will be stored next to secret data, i.e., on the **secret page**
   b. Multiple **probe pages**
2. The client **waits for a deduplication pass** to occur.
3. **Probe**: The client updates its probe pages and measures **how long** it takes for the **server to respond**.
   a. If it was **slow**, then the data was **deduplicated**, so: **secret data = probe data**.
   b. If it was **fast**, then the data remained was **not deduplicated**, so: **secret data ≠ probe data**.

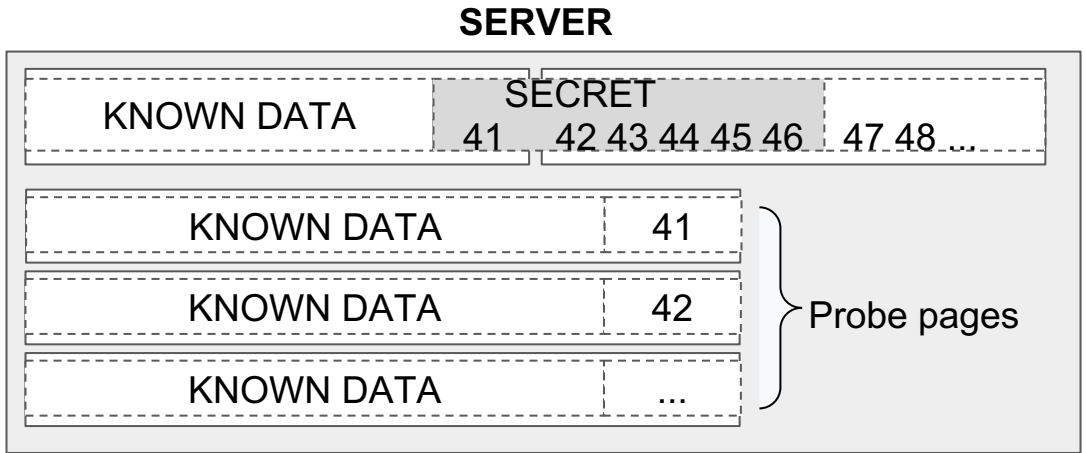
Client-server scenario: Probing

1. **Prime**: The client sends data to the server, including:
   a. Data that will be stored next to secret data, i.e., on the **secret page**
   b. Multiple **probe pages**
2. The client **waits for a deduplication pass** to occur.
3. **Probe**: The client updates its probe pages and measures **how long** it takes for the **server to respond**.
   a. If it was **slow**, then the data was **deduplicated**, so: **secret data = probe data**.
   b. If it was **fast**, then the data remained was **not deduplicated**, so: **secret data ≠ probe data**.

For example, let's **leak one byte** of the server's secret.

Client-server scenario: Probing

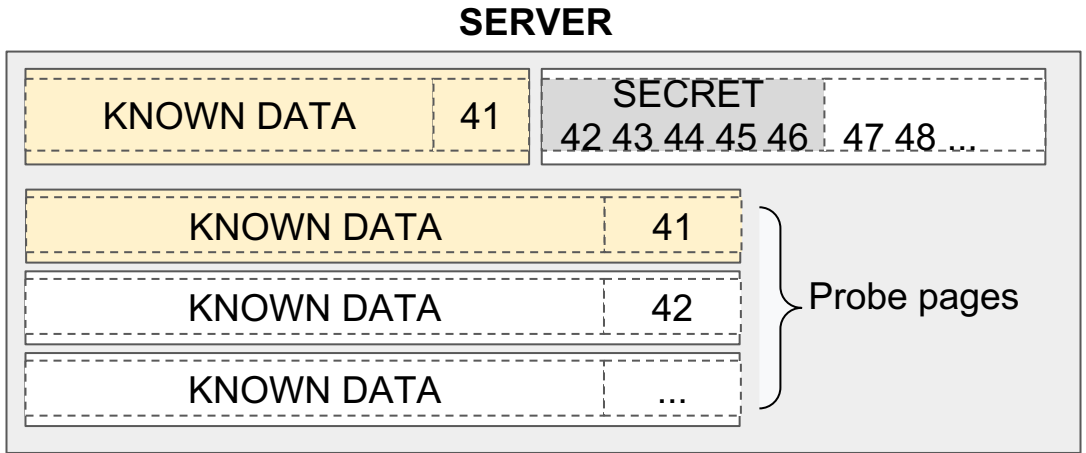
1. **Prime**: The client sends data to the server, including:
   a. Data that will be stored next to secret data, i.e., on the **secret page**
   b. Multiple **probe pages**
2. The client **waits for a deduplication pass** to occur.
3. **Probe**: The client updates its probe pages and measures **how long** it takes for the **server to respond**.
   a. If it was **slow**, then the data was **deduplicated**, so: **secret data = probe data**.
   b. If it was **fast**, then the data remained was **not deduplicated**, so: **secret data ≠ probe data**.



For example, let's **leak one byte** of the server's secret.

Client-server scenario: Probing

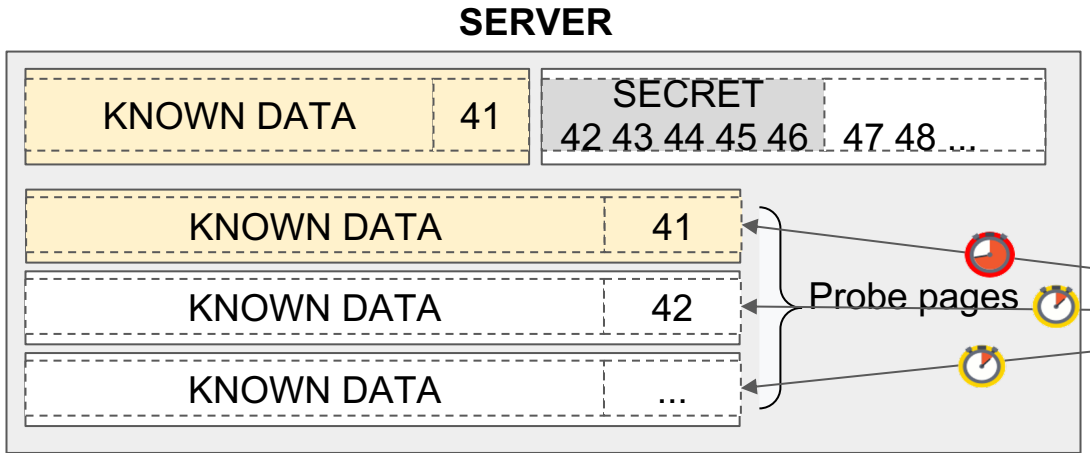
1. **Prime**: The client sends data to the server, including:
   a. Data that will be stored next to secret data, i.e., on the **secret page**
   b. Multiple **probe pages**
2. The client **waits for a deduplication pass** to occur.
3. **Probe**: The client updates its probe pages and measures **how long** it takes for the **server to respond**.
   a. If it was **slow**, then the data was **deduplicated**, so: **secret data = probe data**.
   b. If it was **fast**, then the data remained was **not deduplicated**, so: **secret data ≠ probe data**.



For example, let's **leak one byte** of the server's secret.

Client-server scenario: Probing

1. **Prime**: The client sends data to the server, including:
   a. Data that will be stored next to secret data, i.e., on the **secret page**
   b. Multiple **probe pages**
2. The client **waits for a deduplication pass** to occur.
3. **Probe**: The client updates its probe pages and measures **how long** it takes for the **server to respond**.
   a. If it was **slow**, then the data was **deduplicated**, so: **secret data = probe data**.
   b. If it was **fast**, then the data remained was **not deduplicated**, so: **secret data ≠ probe data**.



For example, let's **leak one byte** of the server's secret.

Client-server scenario: Exploit steps

1. **Prime**: The client sends data to the server, including:
   a. Data that will be stored next to secret data, i.e., on the **secret page**
   b. Multiple **probe pages**
2. The client **waits for a deduplication pass** to occur.
3. **Probe**: The client updates its probe pages and measures **how long** it takes for the **server to respond**.
   a. If it was **slow**, then the data was **deduplicated**, so: **secret data = probe data**.
   b. If it was **fast**, then the data remained was **not deduplicated**, so: **secret data ≠ probe data**.

Client-server scenario: Exploit steps

1. **Prime**: The client sends data to the server, including:
   a. Data that will be stored next to secret data, i.e., on the **secret page**
   b. Multiple **probe pages**
2. The client **waits for a deduplication pass** to occur.
3. **Probe**: The client updates its probe pages and measures **how long** it takes for the **server to respond**.
   a. If it was **slow**, then the data was **deduplicated**, so: **secret data = probe data**.
   b. If it was **fast**, then the data remained was **not deduplicated**, so: **secret data ≠ probe data**.
4. The client repeats steps 1–3 to disclose secret data **byte-by-byte**.

Client-server scenario: Alignment probing

Client-server scenario: Alignment probing

- We assume that the client can manipulate the **alignment of secret data** based on how much data it sends to the server.

Client-server scenario: Alignment probing

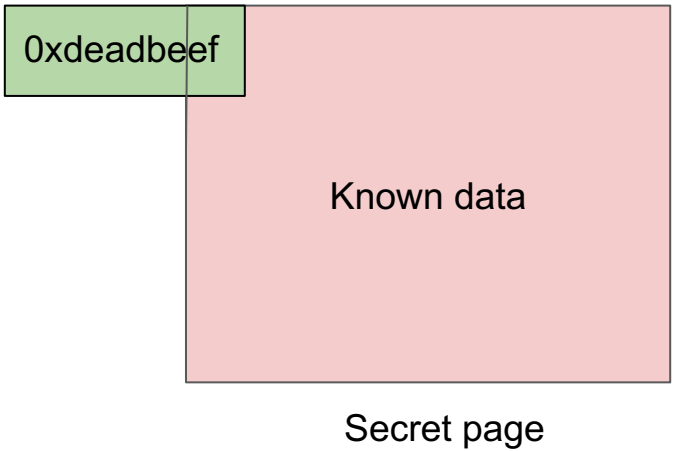
- We assume that the client can manipulate the **alignment of secret data** based on how much data it sends to the server.
- Hence, the client can push the secret data **out of the memory page boundary to reduce entropy**.

Client-server scenario: Alignment probing

- We assume that the client can manipulate the **alignment of secret data** based on how much data it sends to the server.
- Hence, the client can push the secret data **out of the memory page boundary to reduce entropy**.
- As a result, the client can **disclose secret memory byte-by-byte**.

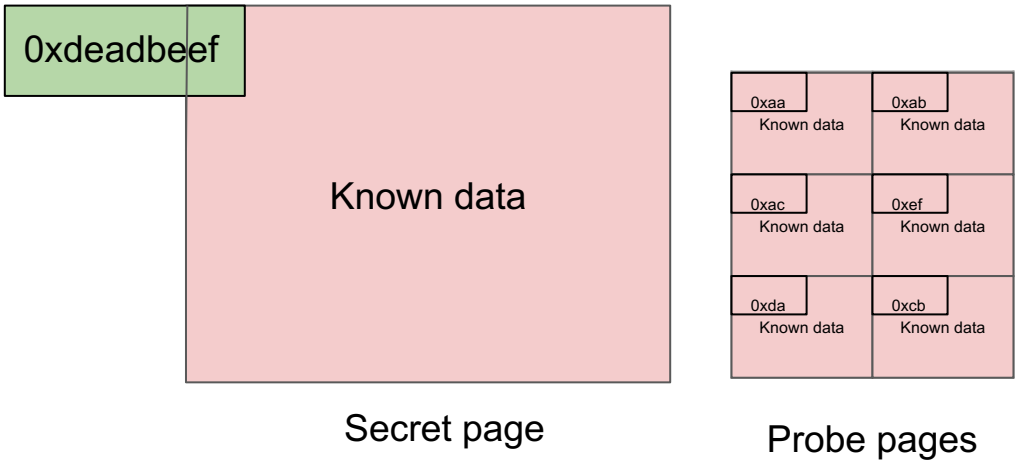
Client-server scenario: Alignment probing

- We assume that the client can manipulate the **alignment of secret data** based on how much data it sends to the server.
- Hence, the client can push the secret data **out of the memory page boundary to reduce entropy**.
- As a result, the client can **disclose secret memory byte-by-byte**.



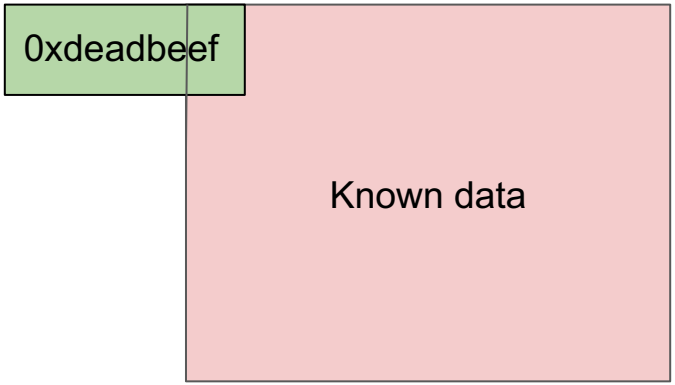
Client-server scenario: Alignment probing

- We assume that the client can manipulate the **alignment of secret data** based on how much data it sends to the server.
- Hence, the client can push the secret data **out of the memory page boundary to reduce entropy**.
- As a result, the client can **disclose secret memory byte-by-byte**.

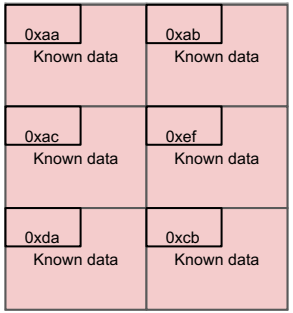


Client-server scenario: Alignment probing

- We assume that the client can manipulate the **alignment of secret data** based on how much data it sends to the server.
- Hence, the client can push the secret data **out of the memory page boundary to reduce entropy**.
- As a result, the client can **disclose secret memory byte-by-byte**.

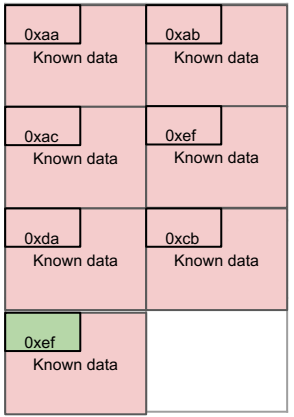


Secret page



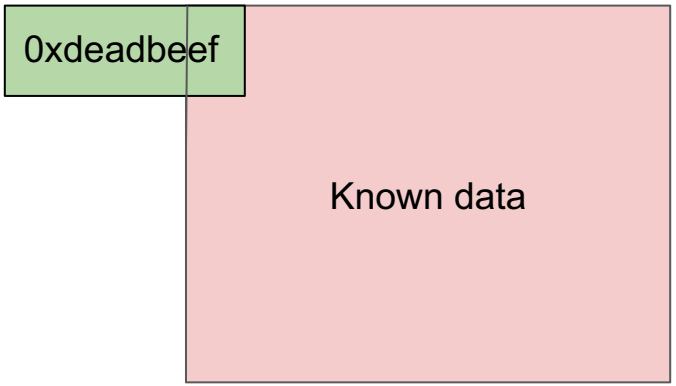
Probe pages

Physical Memory

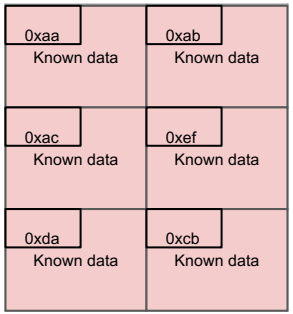


Client-server scenario: Alignment probing

- We assume that the client can manipulate the **alignment of secret data** based on how much data it sends to the server.
- Hence, the client can push the secret data **out of the memory page boundary to reduce entropy**.
- As a result, the client can **disclose secret memory byte-by-byte**.

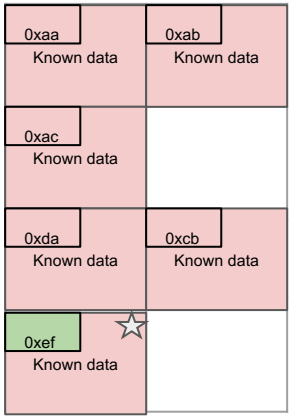


Secret page



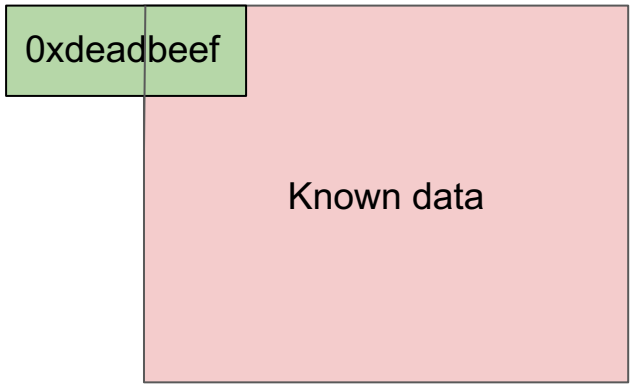
Probe pages

Physical Memory

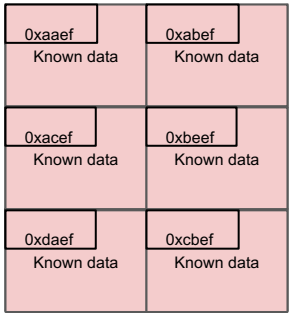


Client-server scenario: Alignment probing

- We assume that the client can manipulate the **alignment of secret data** based on how much data it sends to the server.
- Hence, the client can push the secret data **out of the memory page boundary to reduce entropy**.
- As a result, the client can **disclose secret memory byte-by-byte**.

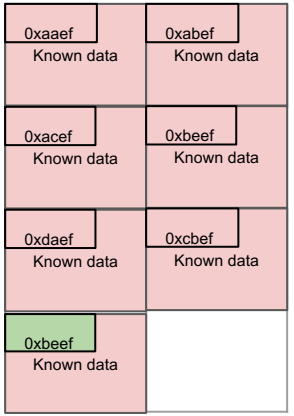


Secret page



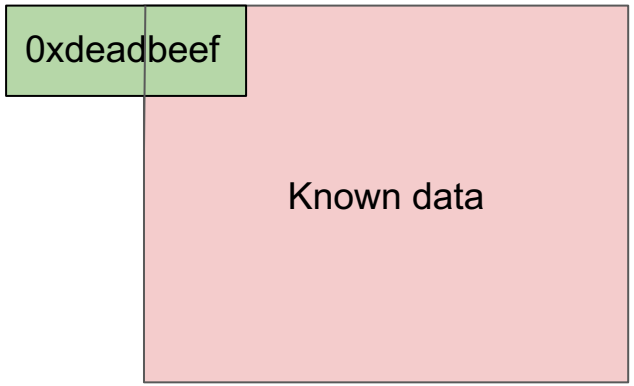
Probe pages

Physical Memory

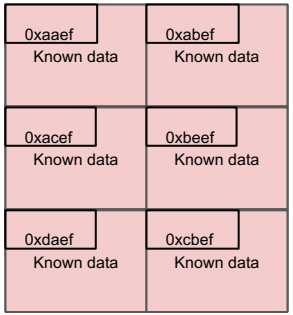


Client-server scenario: Alignment probing

- We assume that the client can manipulate the **alignment of secret data** based on how much data it sends to the server.
- Hence, the client can push the secret data **out of the memory page boundary to reduce entropy**.
- As a result, the client can **disclose secret memory byte-by-byte**.

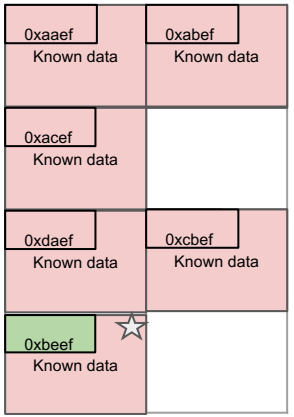


Secret page



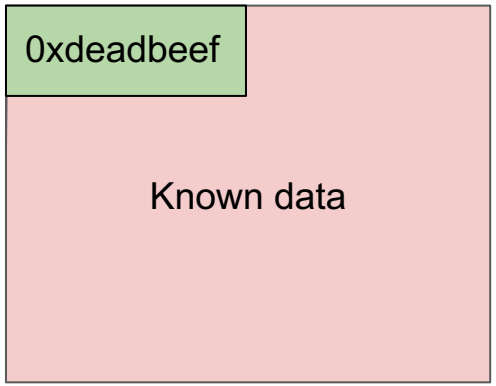
Probe pages

Physical Memory

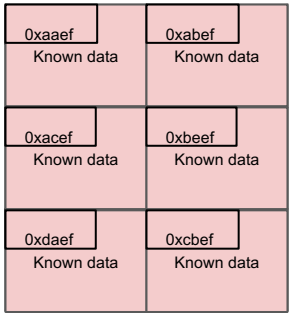


Client-server scenario: Alignment probing

- We assume that the client can manipulate the **alignment of secret data** based on how much data it sends to the server.
- Hence, the client can push the secret data **out of the memory page boundary to reduce entropy**.
- As a result, the client can **disclose secret memory byte-by-byte**.

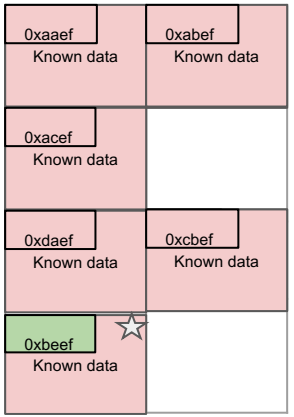


Secret page

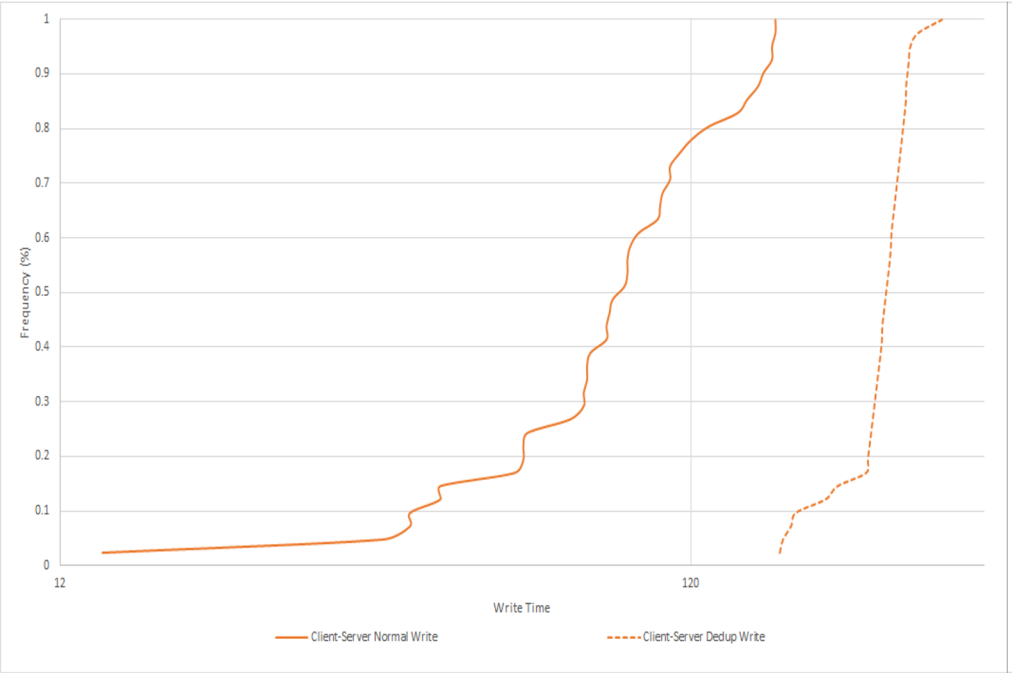


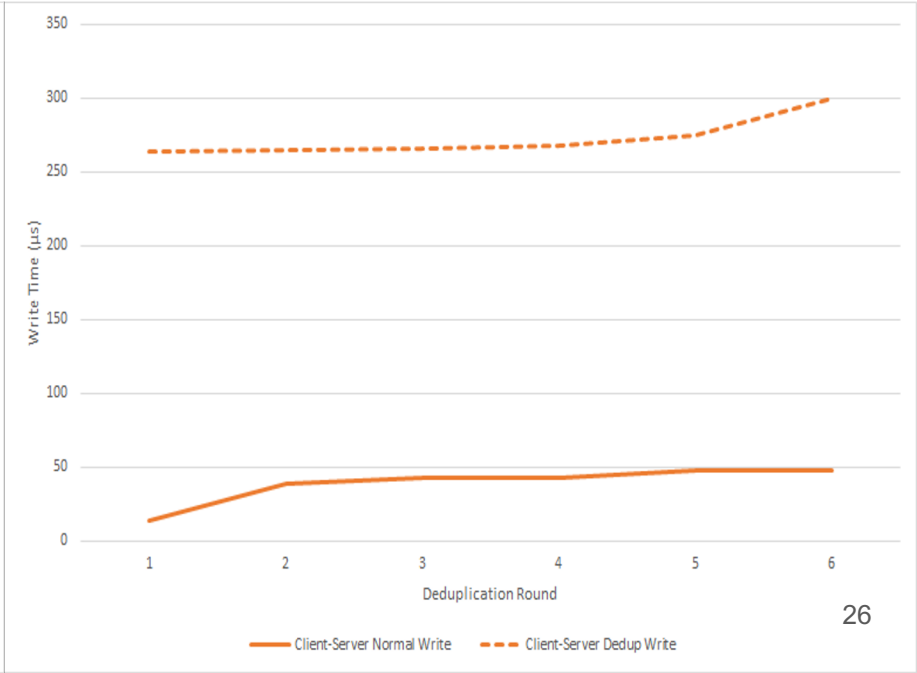
Probe pages

Physical Memory



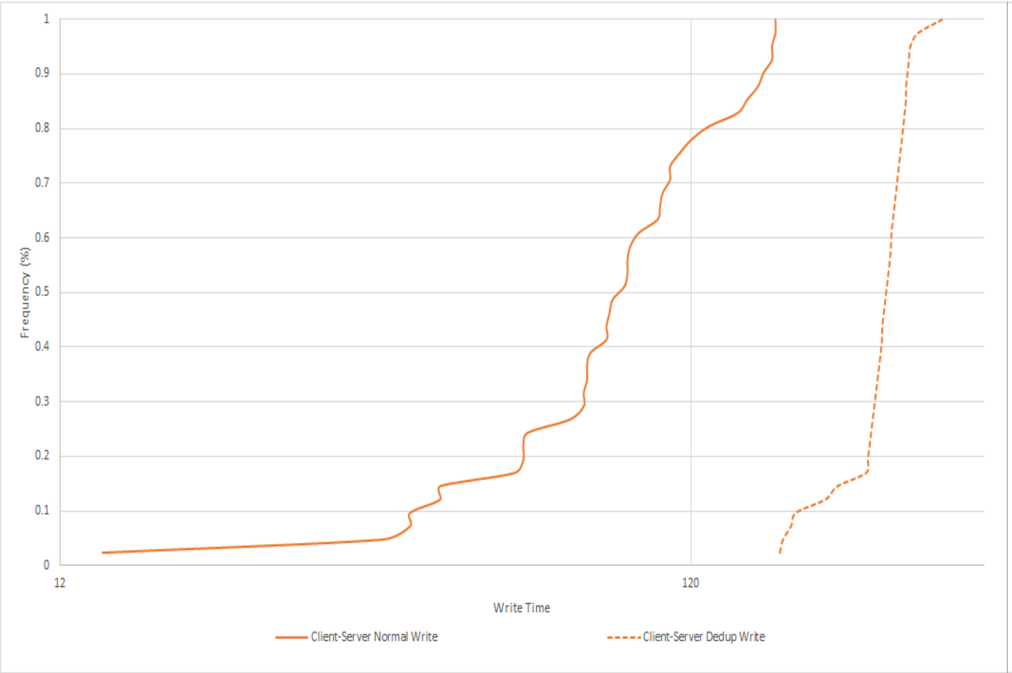
Client-server scenario: Results

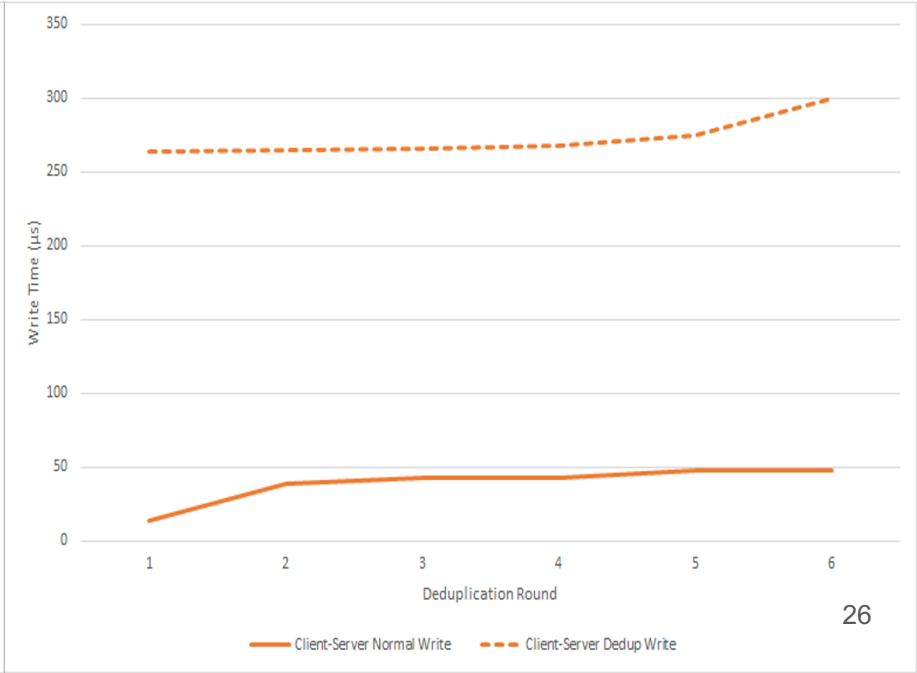




Client-server scenario: Results

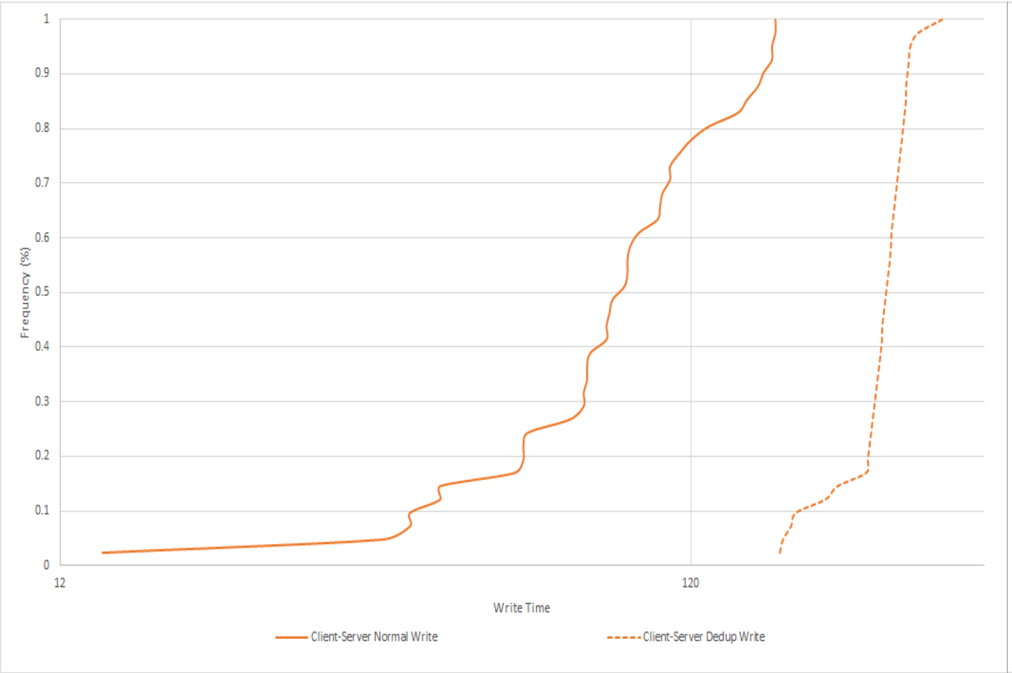
- There are significant timing differences between a normal write and dedup write

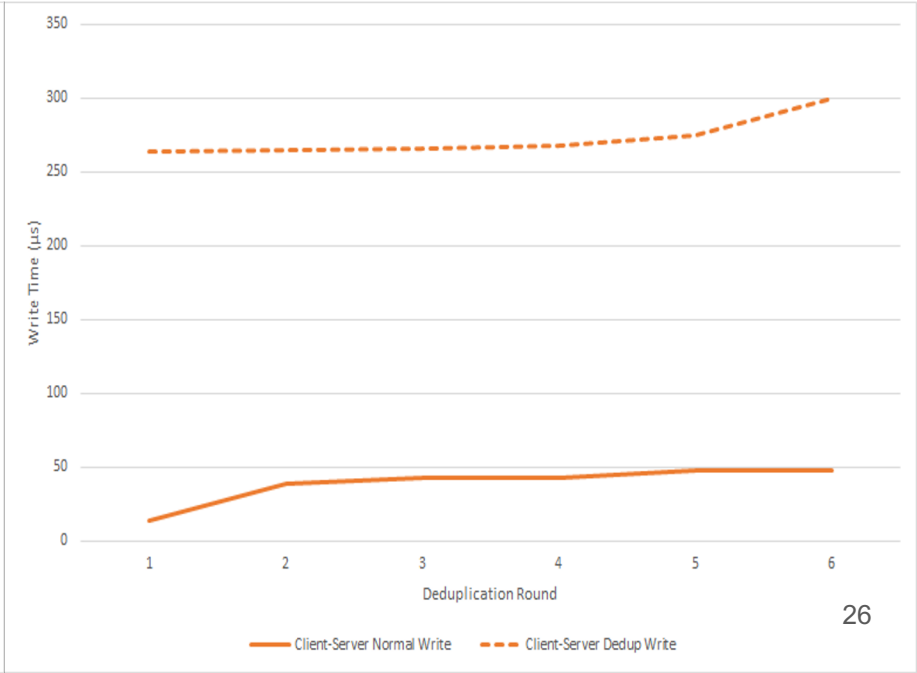




Client-server scenario: Results

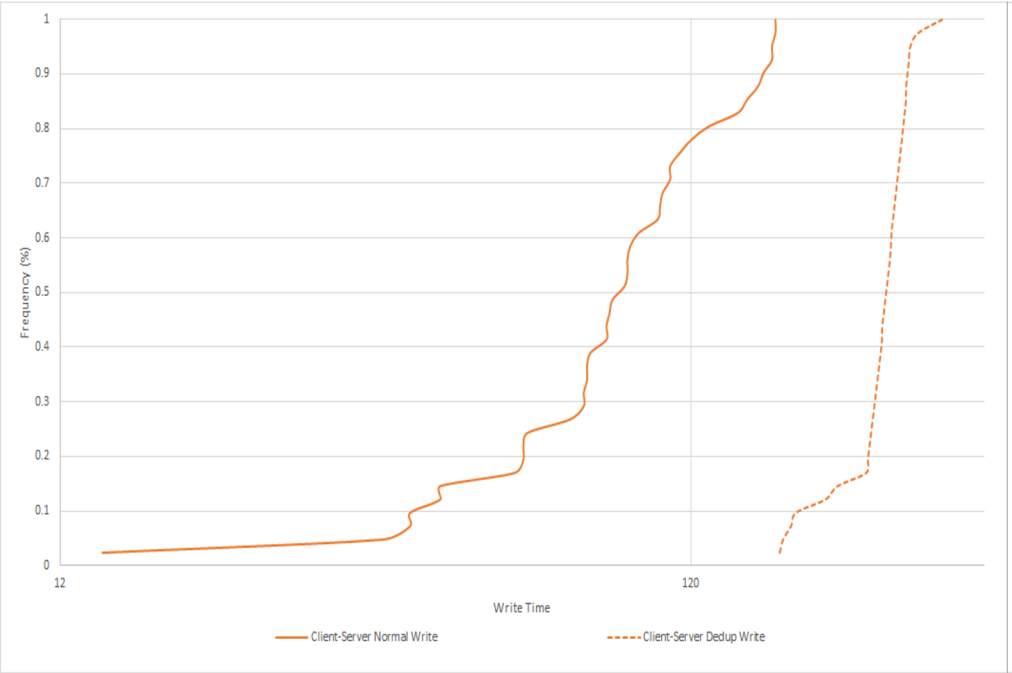
- There are significant timing differences between a normal write and dedup write
  - The average server response time after a **normal write**: 97µs

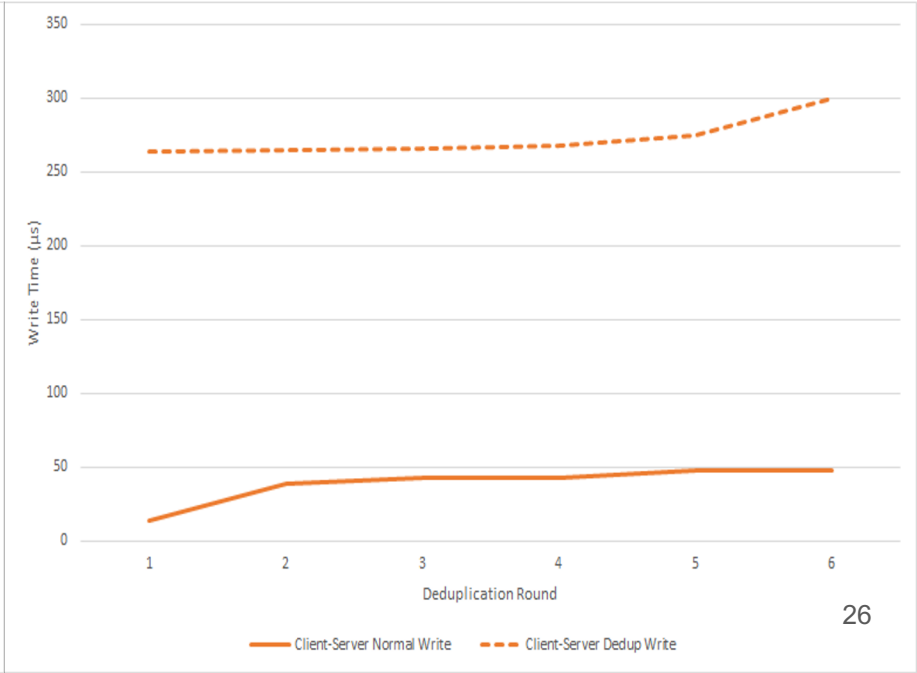




Client-server scenario: Results

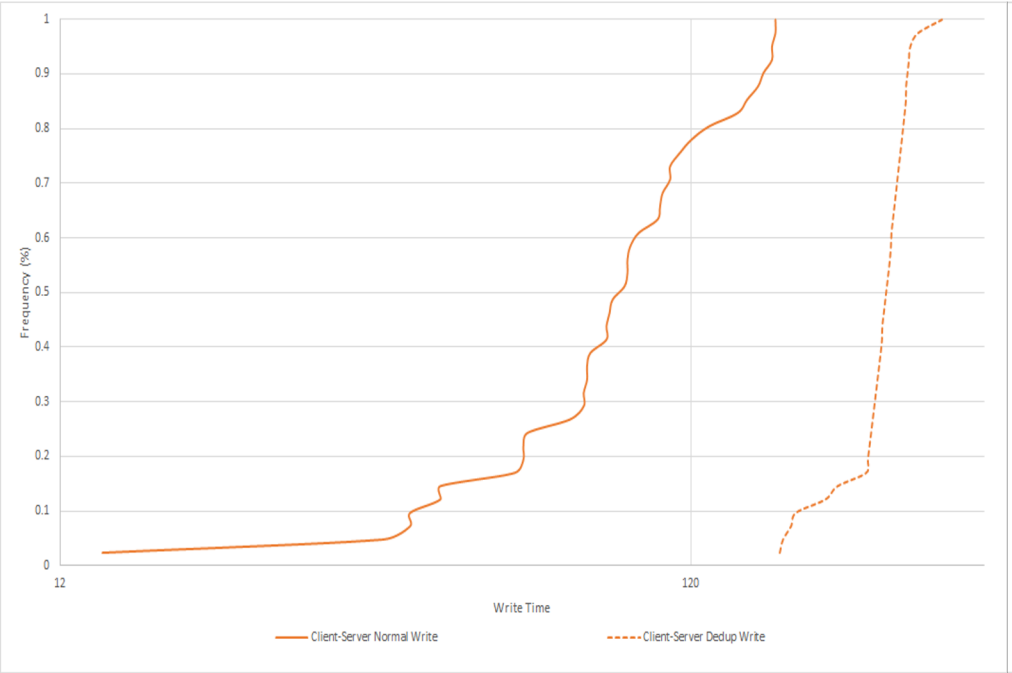
- There are significant timing differences between a normal write and dedup write
  - The average server response time after a **normal write**: 97µs
  - The average server response time after a **CoW write**: 240µs

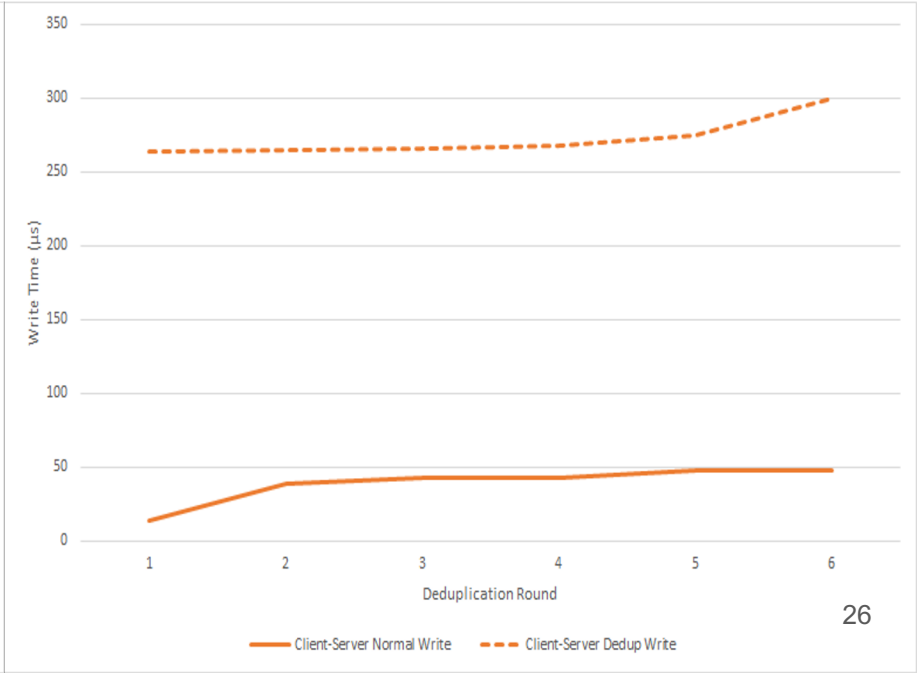




Client-server scenario: Results

- There are significant timing differences between a normal write and dedup write
  - The average server response time after a **normal write**: 97µs
  - The average server response time after a **CoW write**: 240µs
- 85% of the CoW write operations took more than 205µs to complete





Client-server scenario: Conclusion

Client-server scenario: Conclusion

- We did make some **assumptions** (e.g., that we have an alignment probing primitive), but this was meant to be a **proof-of-concept**.

Client-server scenario: Conclusion

- We did make some **assumptions** (e.g., that we have an alignment probing primitive), but this was meant to be a **proof-of-concept**.
- Some programs (e.g., a client-server scenario) **necessarily handle both untrusted data and trusted data**.

Client-server scenario: Conclusion

- We did make some **assumptions** (e.g., that we have an alignment probing primitive), but this was meant to be a **proof-of-concept**.
- Some programs (e.g., a client-server scenario) **necessarily handle both untrusted data and trusted data**.
- Hence, such programs are **not easily amenable to security domain-based deduplication mitigations**, which require the separation of trusted and untrusted data.

Browser cross-tab scenario: Overview

Browser cross-tab scenario: Overview

- In this scenario, **the browser is the victim** and a **webpage is the attacker**.

Browser cross-tab scenario: Overview

- In this scenario, **the browser is the victim** and a **webpage is the attacker**.
  - The **browser** has **multiple tabs open**

Browser cross-tab scenario: Overview

- In this scenario, **the browser is the victim** and a **webpage is the attacker**.
  - The **browser** has **multiple tabs open**
  - The **malicious webpage** is trying to deduce **which other tabs are open**

Browser cross-tab scenario: Overview

- In this scenario, **the browser is the victim** and a **webpage is the attacker**.
  - The **browser** has **multiple tabs open**
  - The **malicious webpage** is trying to deduce **which other tabs are open**
- We exploit Firefox v83.0's **partial implementation of site isolation**, i.e.:

Browser cross-tab scenario: Overview

- In this scenario, **the browser is the victim** and a **webpage is the attacker**.
  - The **browser** has **multiple tabs open**
  - The **malicious webpage** is trying to deduce **which other tabs are open**
- We exploit Firefox v83.0's **partial implementation of site isolation**, i.e.:
  - Firefox uses a **maximum of 8 content processes**, which run e.g., tabs.

Browser cross-tab scenario: Overview

- In this scenario, **the browser is the victim** and a **webpage is the attacker**.
  - The **browser** has **multiple tabs open**
  - The **malicious webpage** is trying to deduce **which other tabs are open**
- We exploit Firefox v83.0's **partial implementation of site isolation**, i.e.:
  - Firefox uses a **maximum of 8 content processes**, which run e.g., tabs.
  - If a **9th tab** is opened, then Firefox runs it in the **same process as another tab**.

Browser cross-tab scenario: Overview

- In this scenario, **the browser is the victim** and a **webpage is the attacker**.
  - The **browser** has **multiple tabs open**
  - The **malicious webpage** is trying to deduce **which other tabs are open**
- We exploit Firefox v83.0's **partial implementation of site isolation**, i.e.:
  - Firefox uses a **maximum of 8 content processes**, which run e.g., tabs.
  - If a **9th tab** is opened, then Firefox runs it in the **same process as another tab**.
  - Since the **attacker and victim share a process**, their data can deduplicate.

Browser cross-tab scenario: Exploit steps

Browser cross-tab scenario: Exploit steps

1. The browser has at least **9 tabs open**, where at least **1 tab is a malicious webpage**.

Browser cross-tab scenario: Exploit steps

1. The browser has at least **9 tabs open**, where at least **1 tab is a malicious webpage**.
    a. Each victim webpage has some memory fingerprint that is known by the attacker beforehand.

Browser cross-tab scenario: Exploit steps

1. The browser has at least **9 tabs open**, where at least **1 tab is a malicious webpage**.
    a. Each victim webpage has some memory fingerprint that is known by the attacker beforehand.
    b. E.g., “google.com has this page in memory”, “facebook.com has this page in memory”, etc.

Browser cross-tab scenario: Exploit steps

1. The browser has at least **9 tabs open**, where at least **1 tab is a malicious webpage**.
    a. Each victim webpage has some memory fingerprint that is known by the attacker beforehand.
    b. E.g., “google.com has this page in memory”, “facebook.com has this page in memory”, etc.
2. Because of Firefox’s limitation, **one process** runs both the **malicious webpage’s code** and a **victim webpage’s code**.

Browser cross-tab scenario: Exploit steps

1. The browser has at least **9 tabs open**, where at least **1 tab is a malicious webpage**.
    a. Each victim webpage has some memory fingerprint that is known by the attacker beforehand.
    b. E.g., “google.com has this page in memory”, “facebook.com has this page in memory”, etc.
2. Because of Firefox’s limitation, **one process** runs both the **malicious webpage’s code** and a **victim webpage’s code**.
3. The attacker keeps the **fingerprints** of possible victim webpages **in its memory**.

Browser cross-tab scenario: Priming

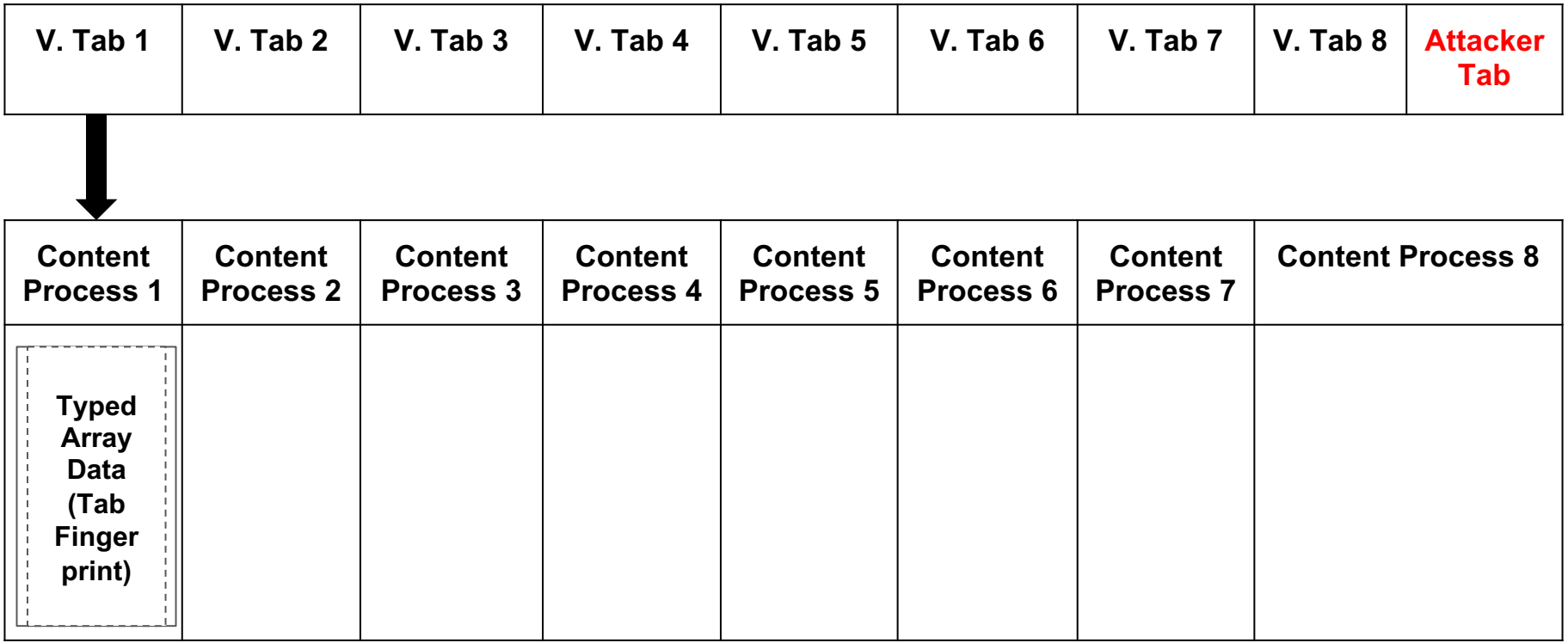
V. Tab 1	V. Tab 2	V. Tab 3	V. Tab 4	V. Tab 5	V. Tab 6	V. Tab 7	V. Tab 8	Attacker Tab

Browser cross-tab scenario: Priming

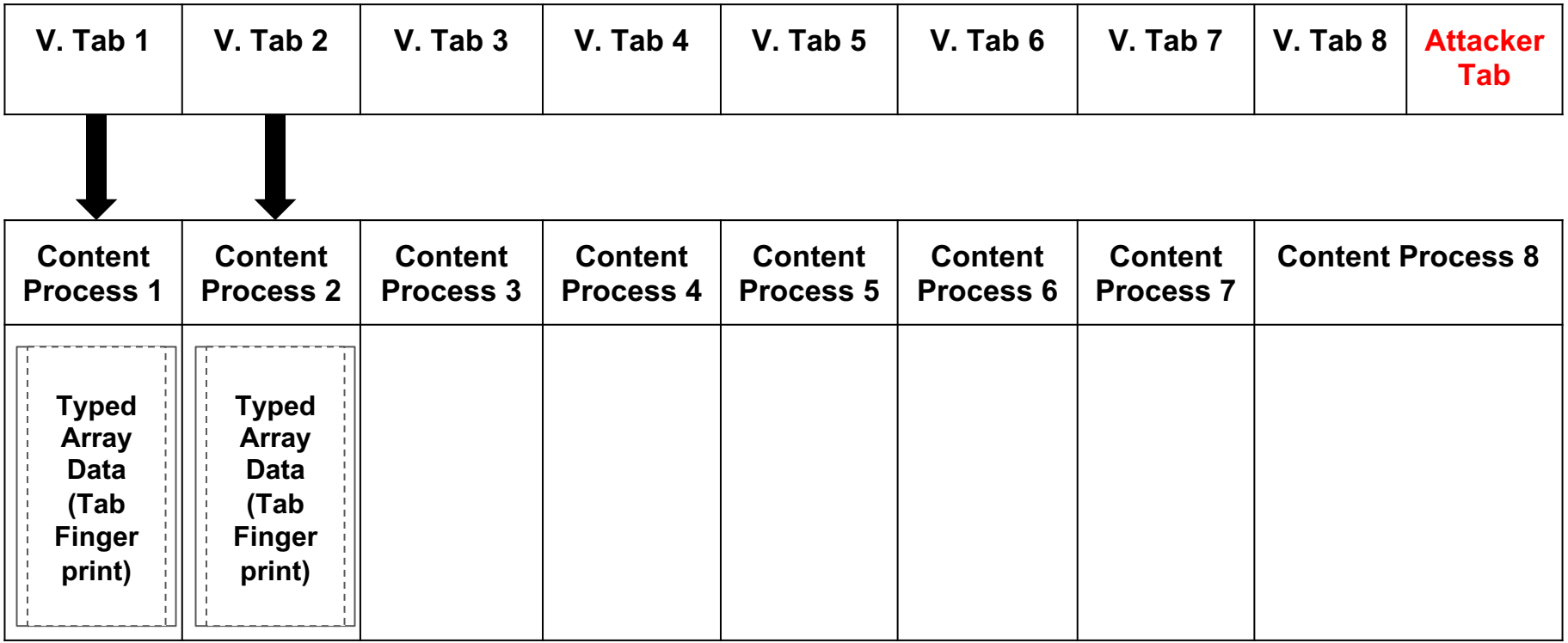
V. Tab 1	V. Tab 2	V. Tab 3	V. Tab 4	V. Tab 5	V. Tab 6	V. Tab 7	V. Tab 8	Attacker Tab

Content Process 1	Content Process 2	Content Process 3	Content Process 4	Content Process 5	Content Process 6	Content Process 7	Content Process 8

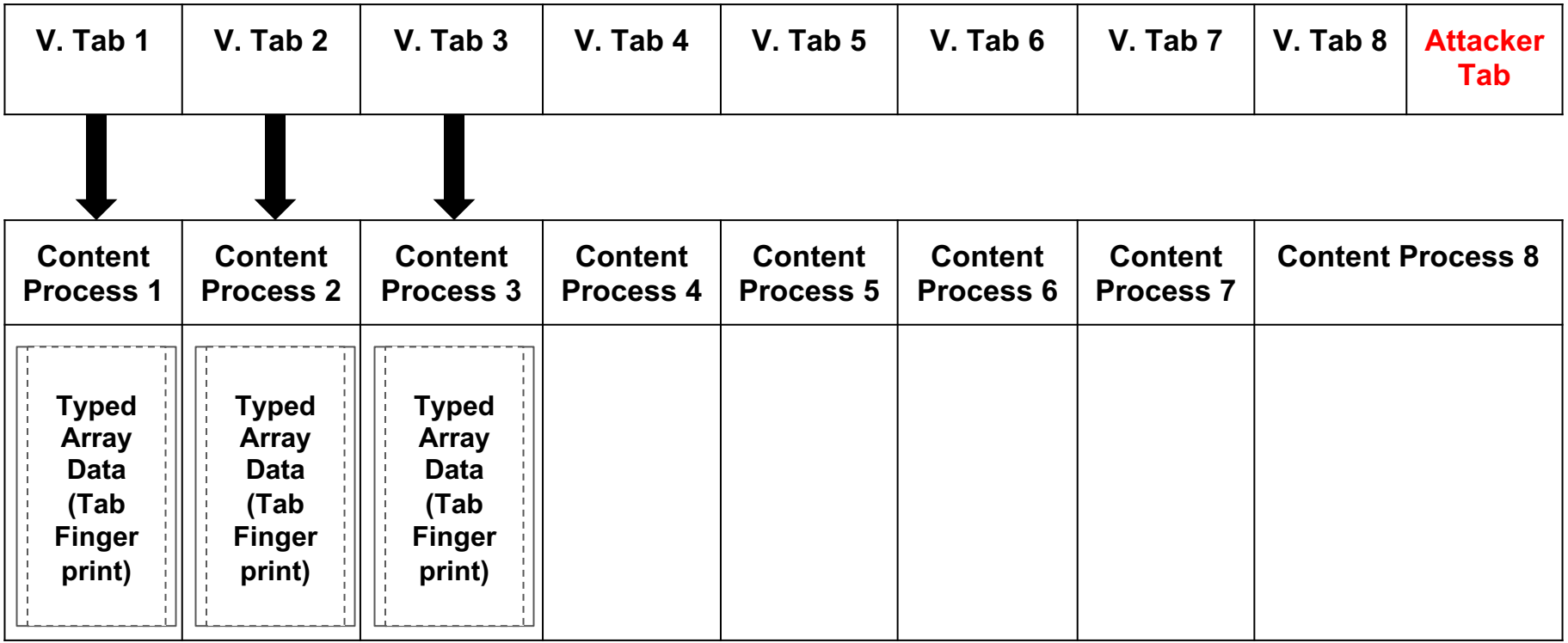
Browser cross-tab scenario: Priming



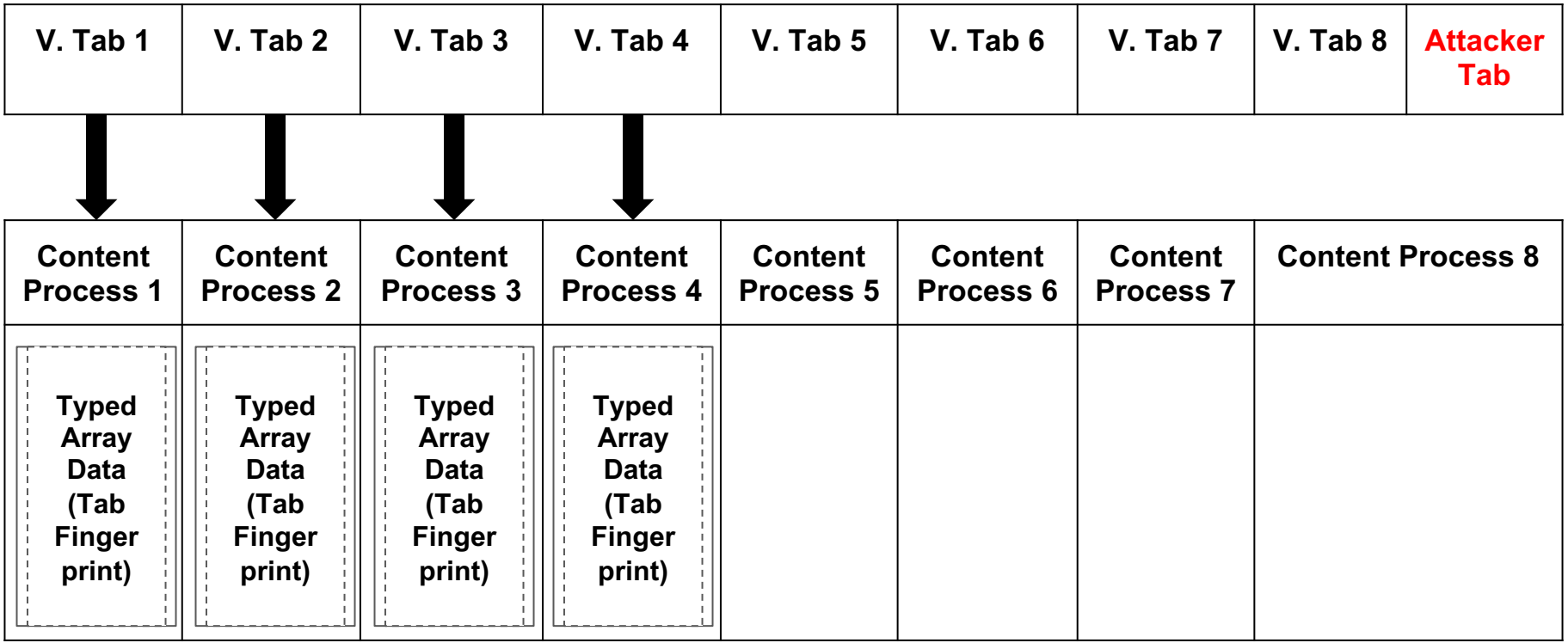
Browser cross-tab scenario: Priming



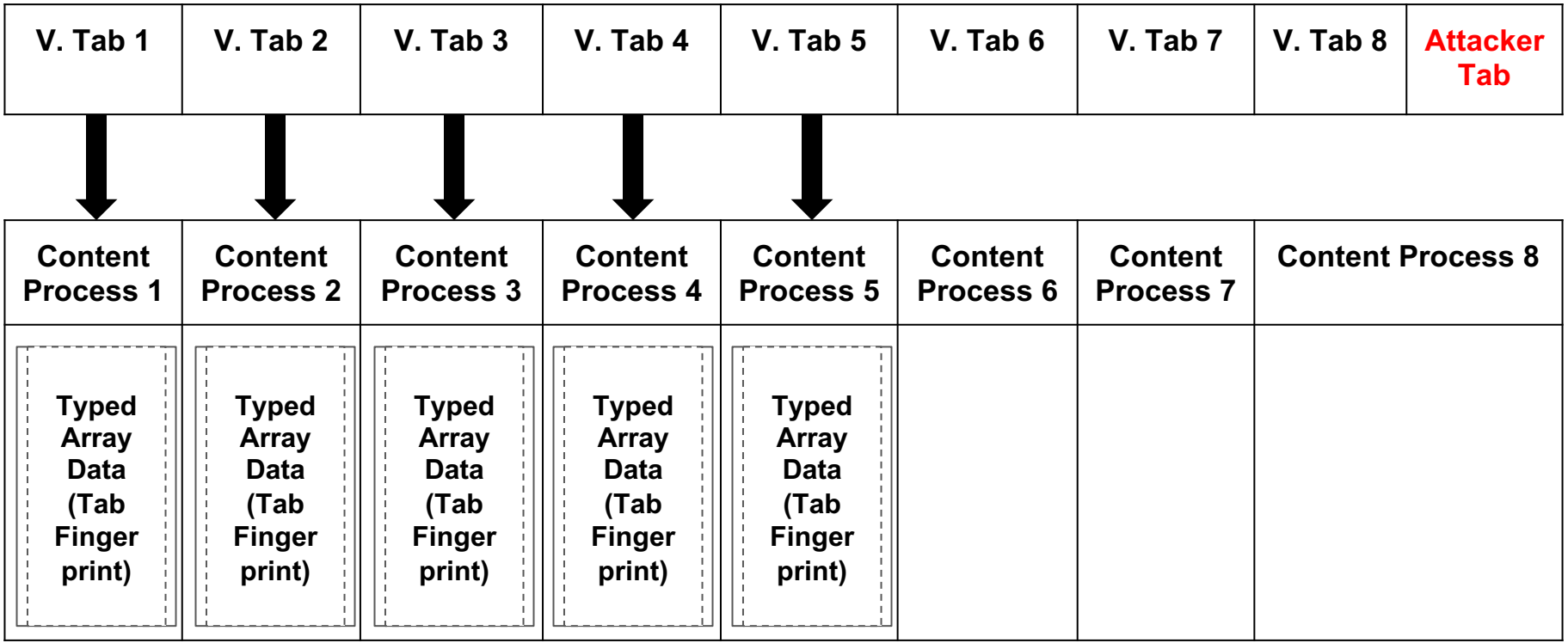
Browser cross-tab scenario: Priming



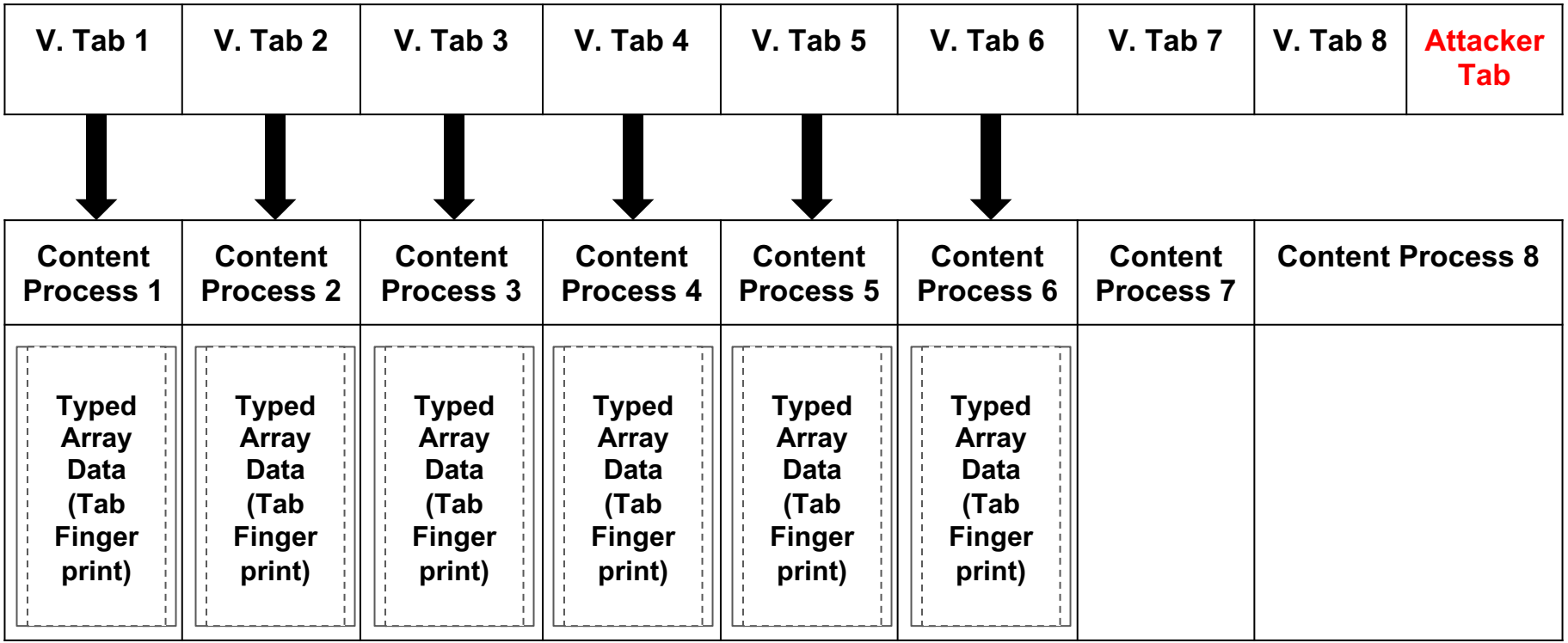
Browser cross-tab scenario: Priming



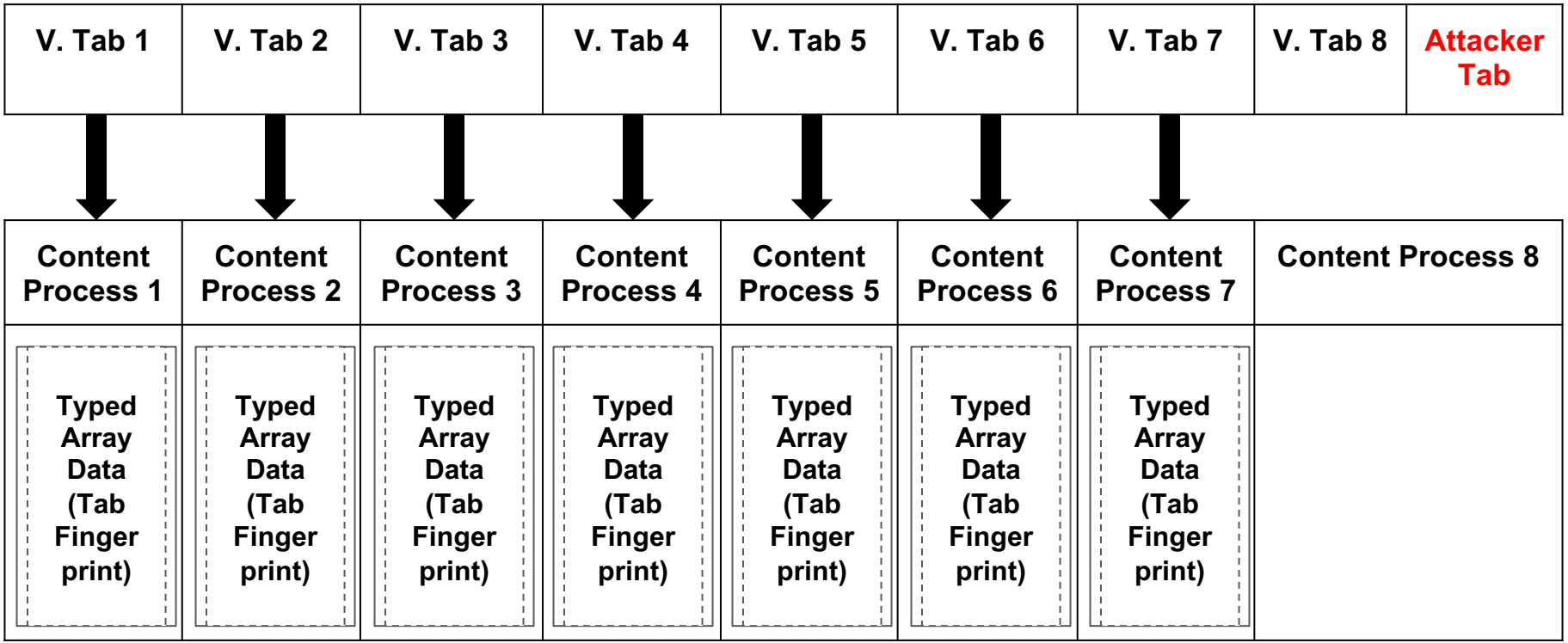
Browser cross-tab scenario: Priming



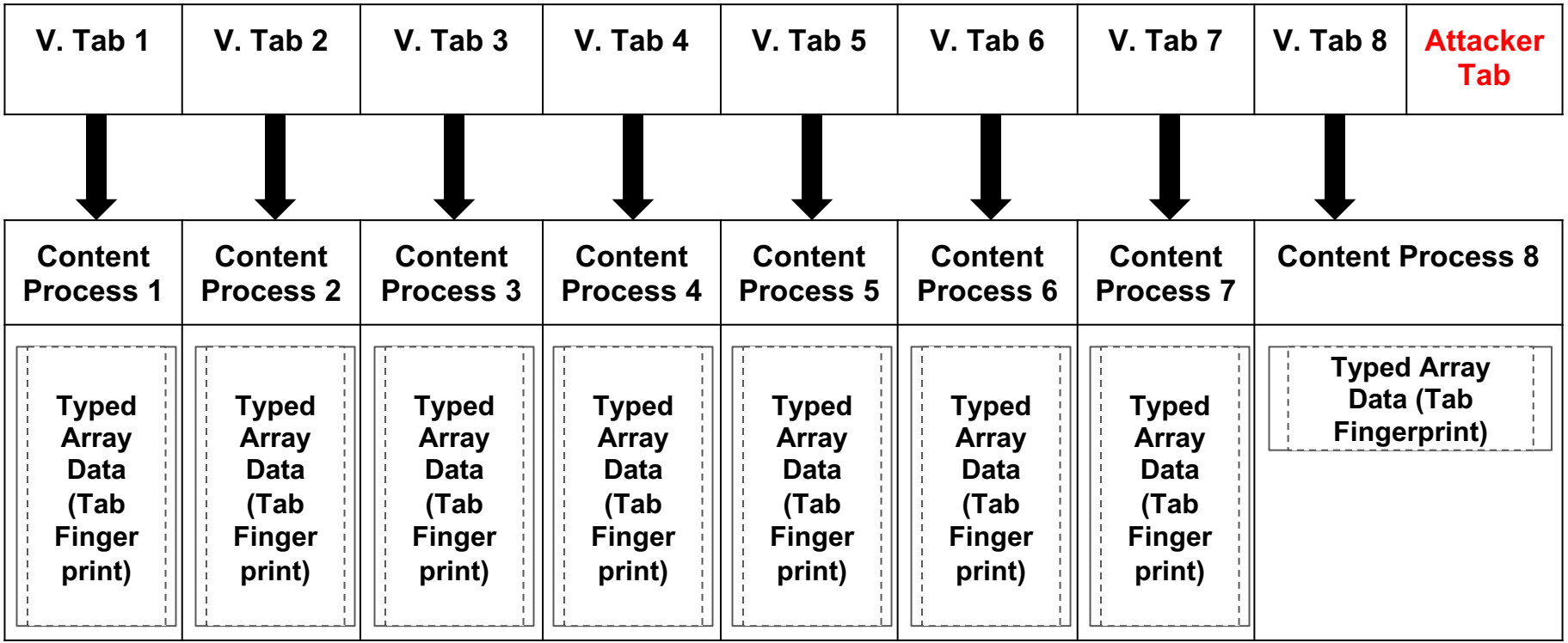
Browser cross-tab scenario: Priming



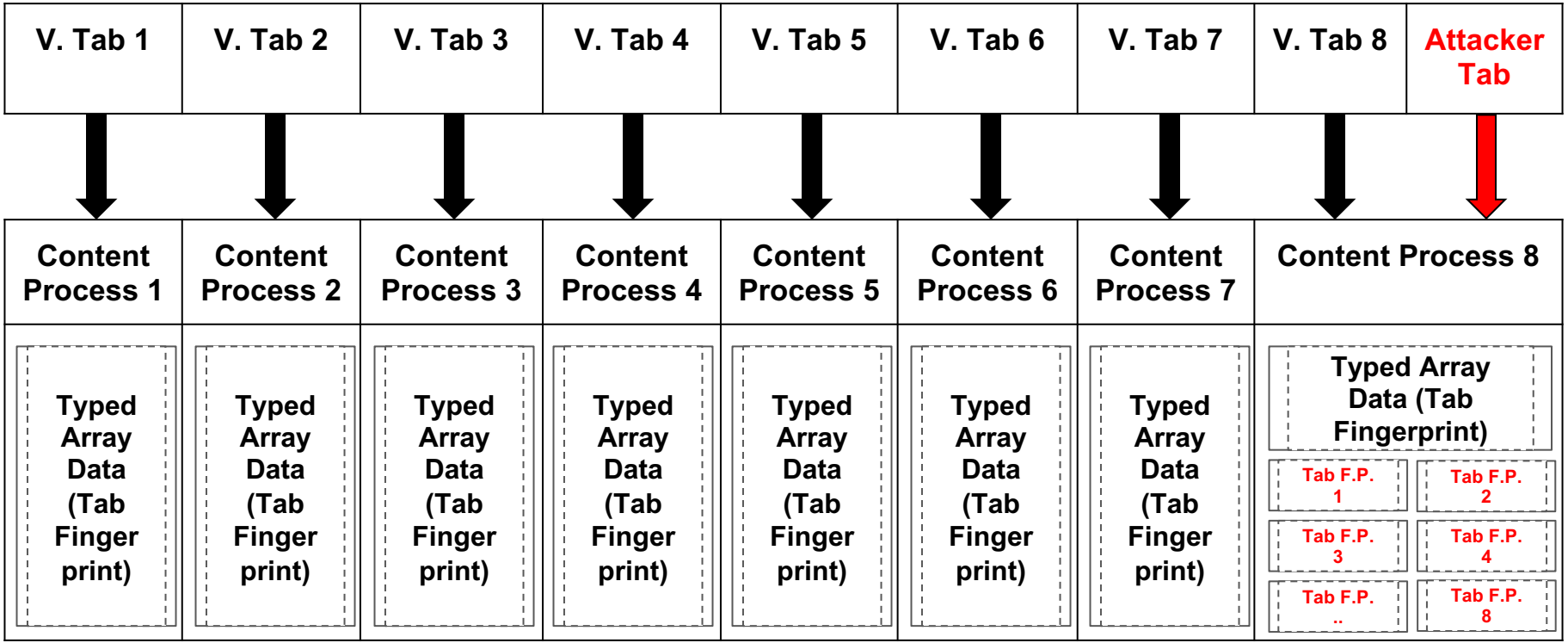
Browser cross-tab scenario: Priming



Browser cross-tab scenario: Priming



Browser cross-tab scenario: Priming



Browser cross-tab scenario: Exploit steps

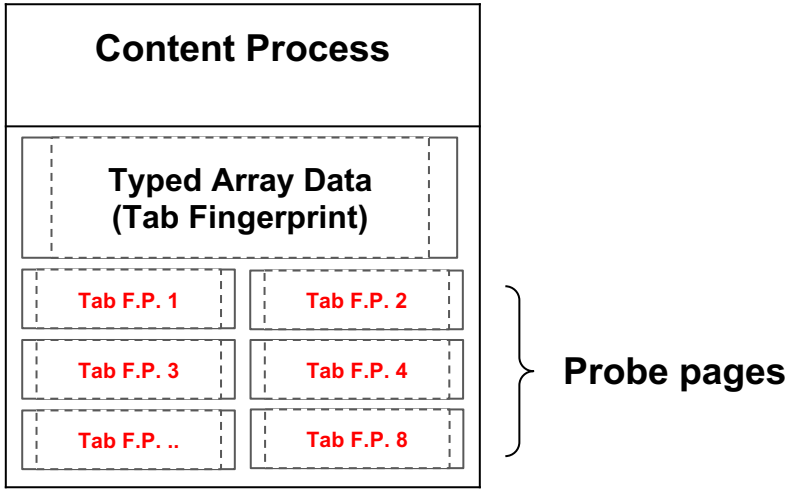
1. The browser has at least **9 tabs open**, where at least **1 tab is a malicious webpage**.
    a. Each victim webpage has some memory fingerprint that is known by the attacker beforehand.
    b. E.g., “google.com has this page in memory”, “facebook.com has this page in memory”, etc.
2. Because of Firefox’s limitation, **one process** runs both the **malicious webpage’s code** and a **victim webpage’s code**.
3. The attacker replicates the **fingerprints** of possible victim webpages in its memory.

Browser cross-tab scenario: Exploit steps

1. The browser has at least **9 tabs open**, where at least **1 tab is a malicious webpage**.
    a. Each victim webpage has some memory fingerprint that is known by the attacker beforehand.
    b. E.g., “google.com has this page in memory”, “facebook.com has this page in memory”, etc.
2. Because of Firefox’s limitation, **one process** runs both the **malicious webpage’s code** and a **victim webpage’s code**.
3. The attacker replicates the **fingerprints** of possible victim webpages in its memory.
4. After a **deduplication pass**, the attacker **writes** to each of its fingerprints, **timing** each operation via a **SharedArrayBuffer counter**.

Browser cross-tab scenario: Probing

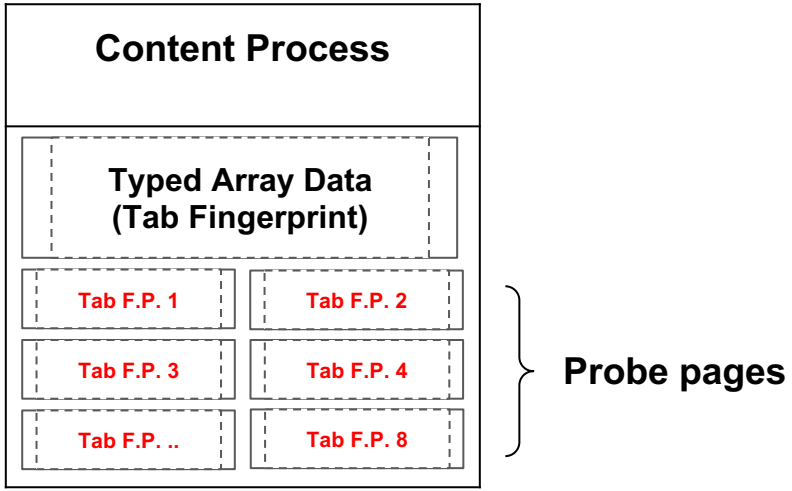
Attacker Tab



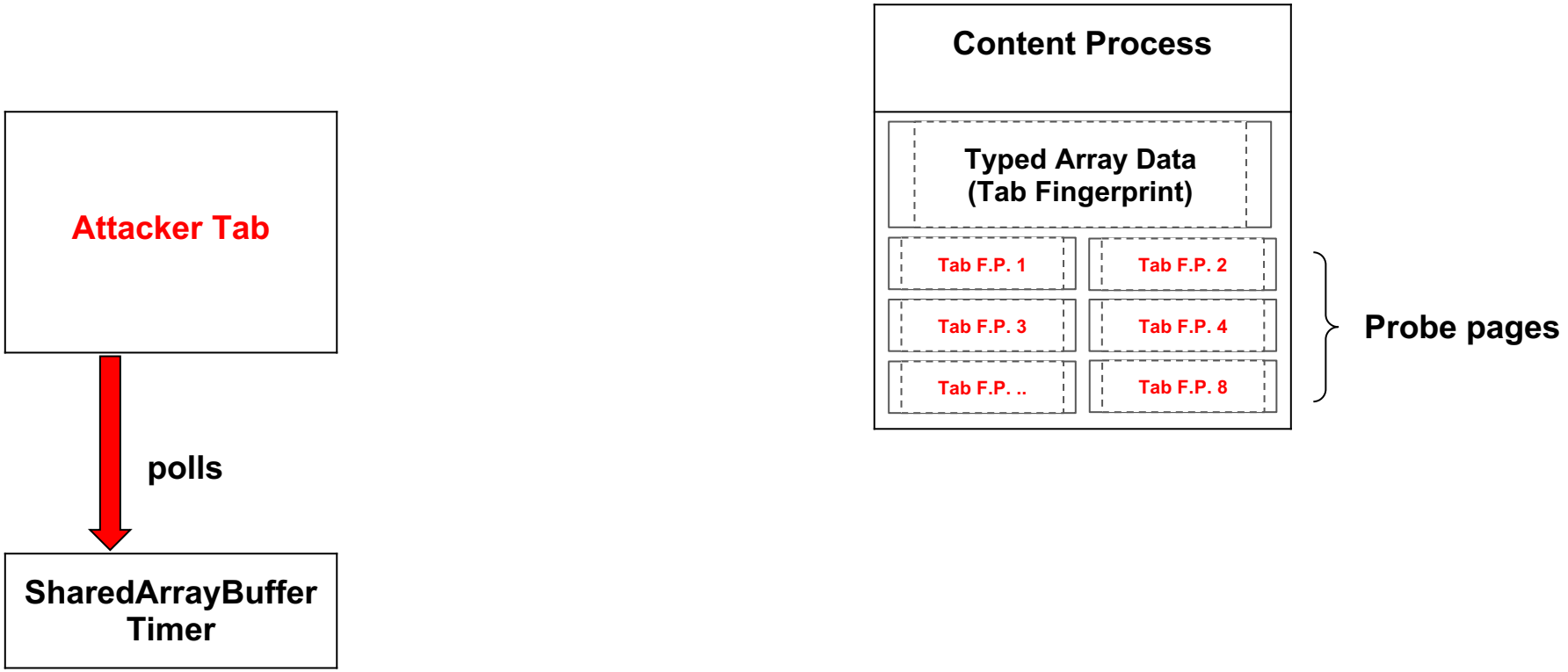
Browser cross-tab scenario: Probing

Attacker Tab

SharedArrayBuffer
Timer



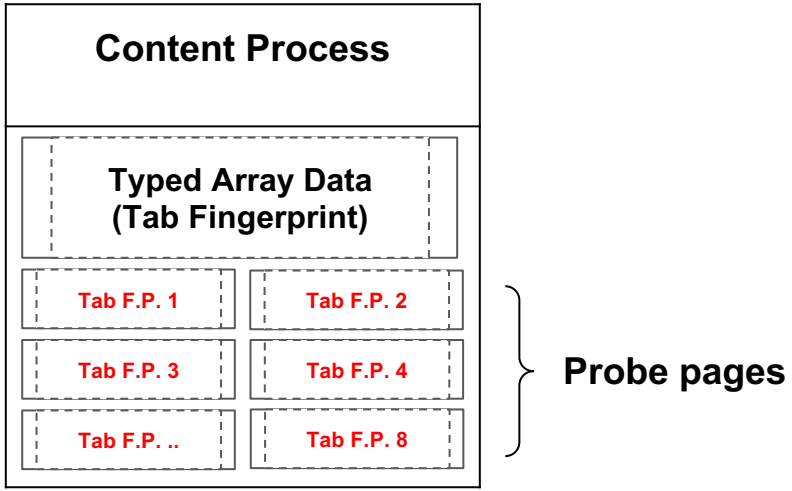
Browser cross-tab scenario: Probing



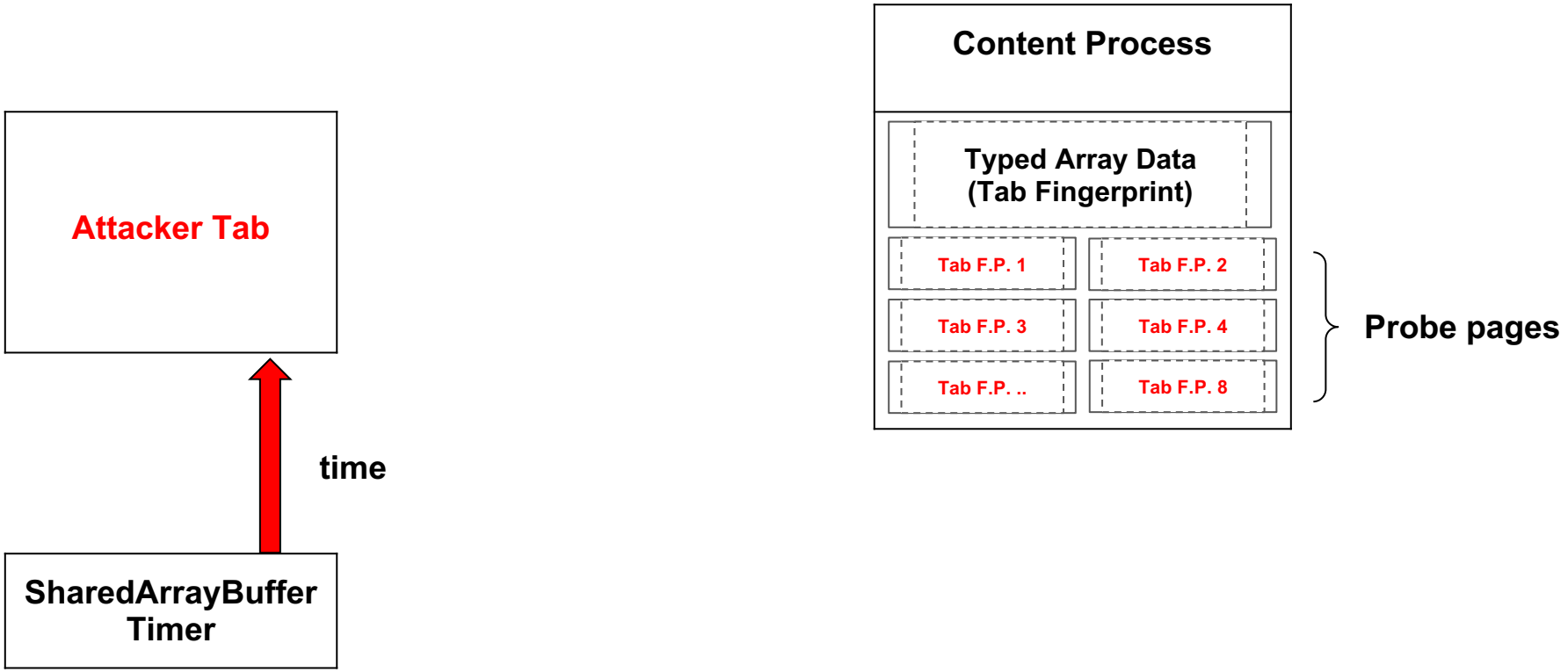
Browser cross-tab scenario: Probing

Attacker Tab

SharedArrayBuffer
Timer



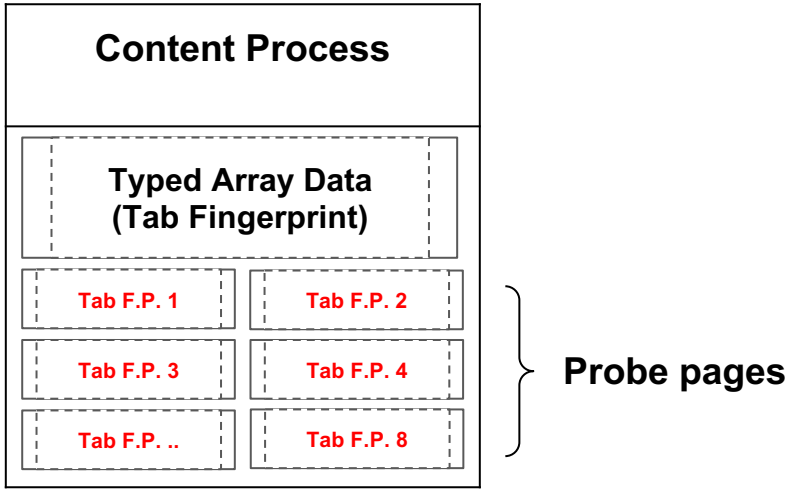
Browser cross-tab scenario: Probing



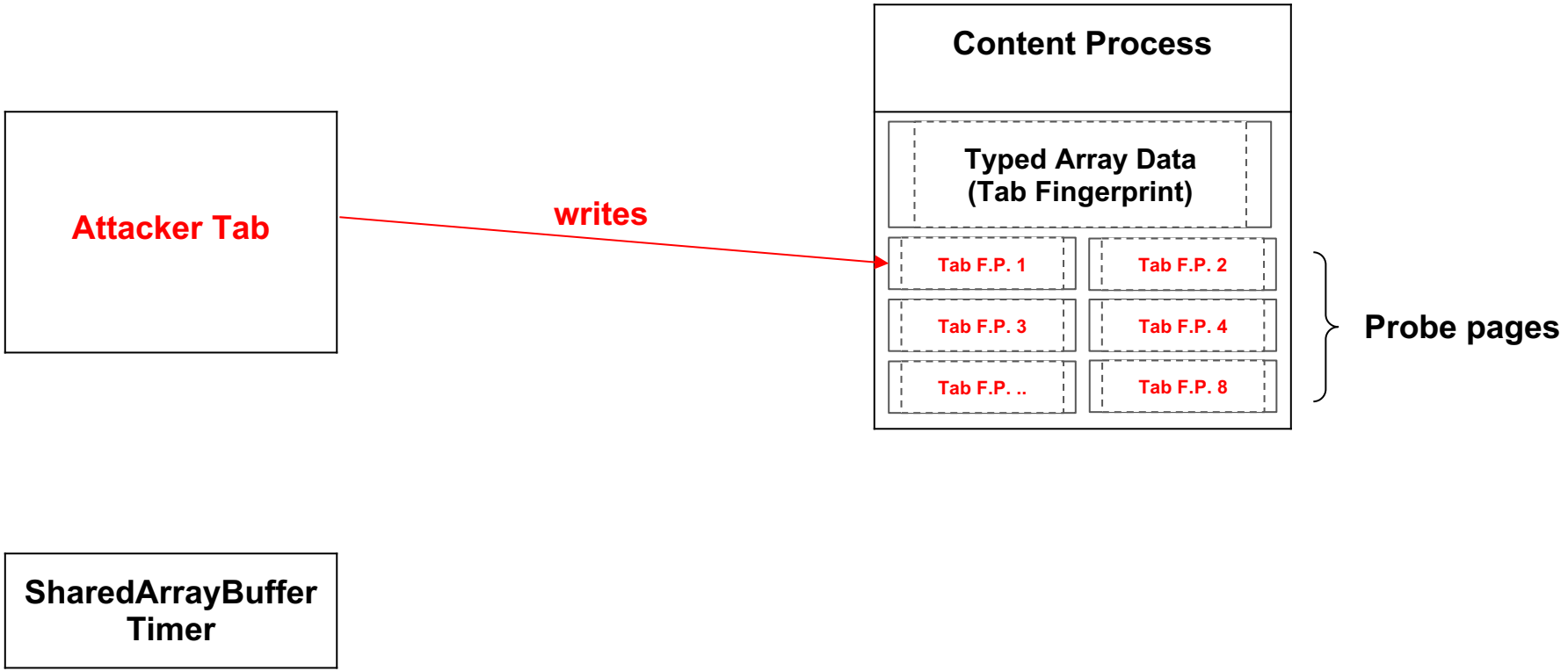
Browser cross-tab scenario: Probing

Attacker Tab

SharedArrayBuffer
Timer



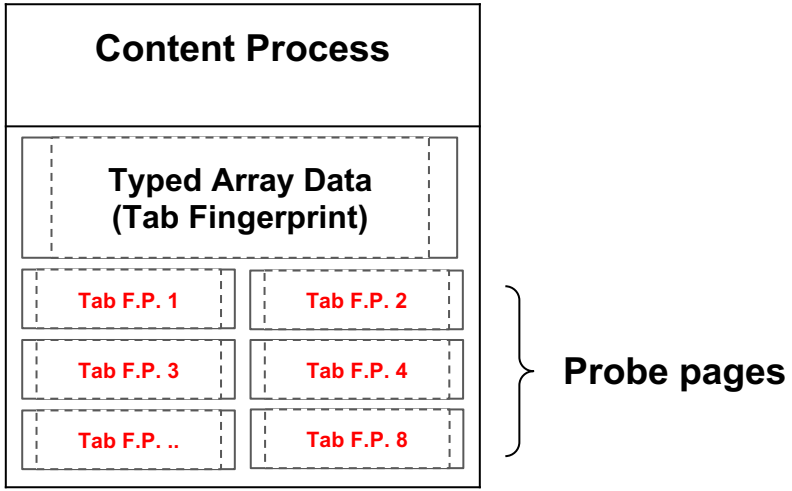
Browser cross-tab scenario: Probing



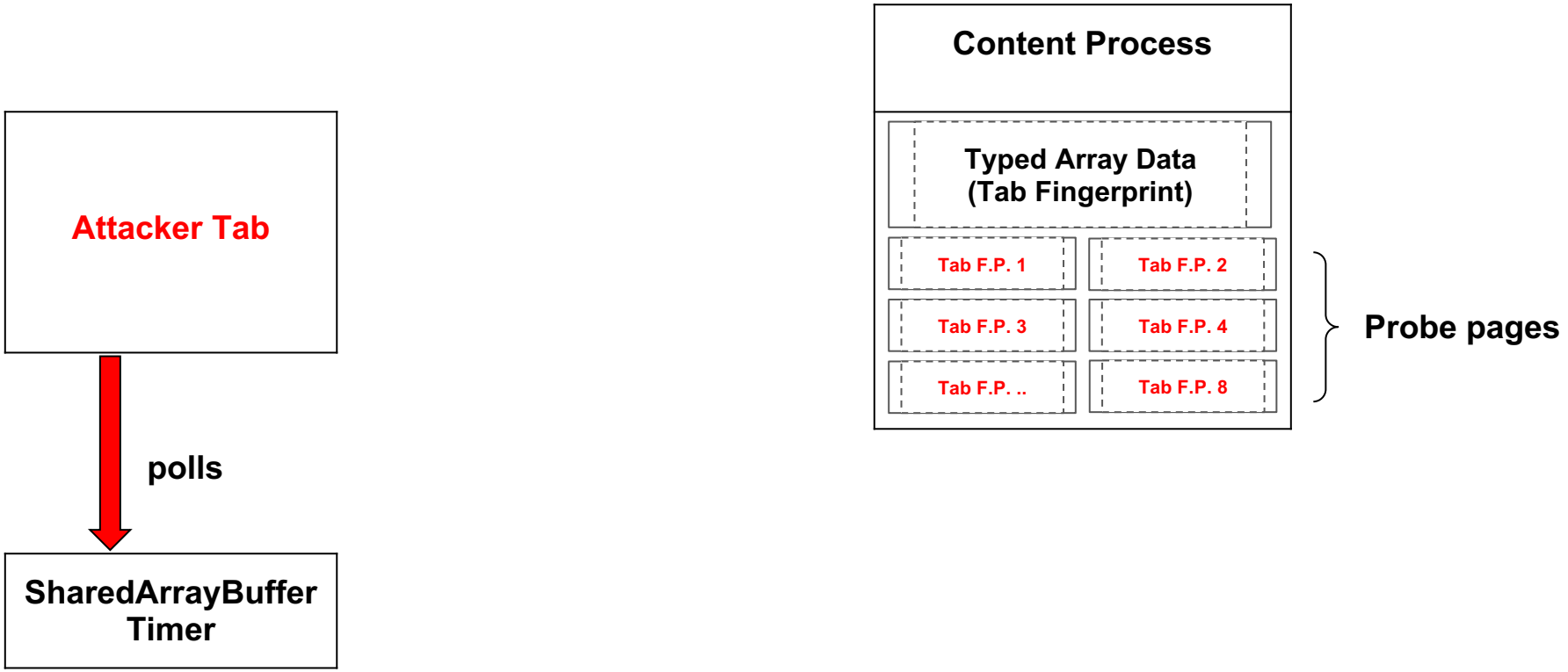
Browser cross-tab scenario: Probing

Attacker Tab

SharedArrayBuffer
Timer



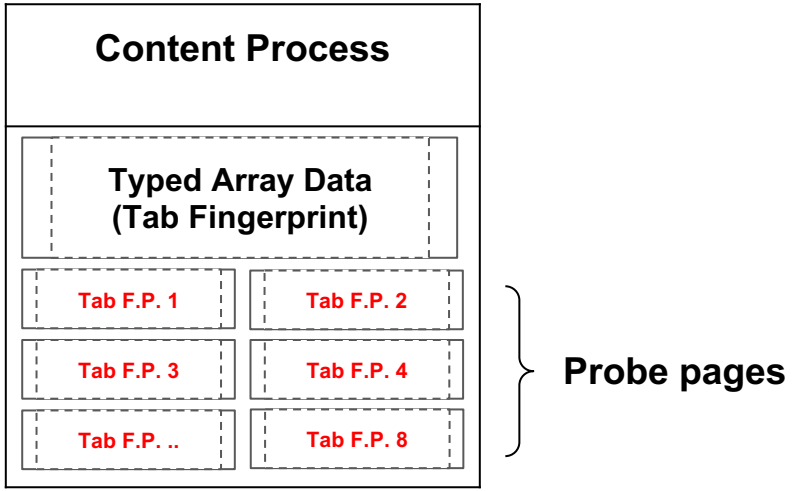
Browser cross-tab scenario: Probing



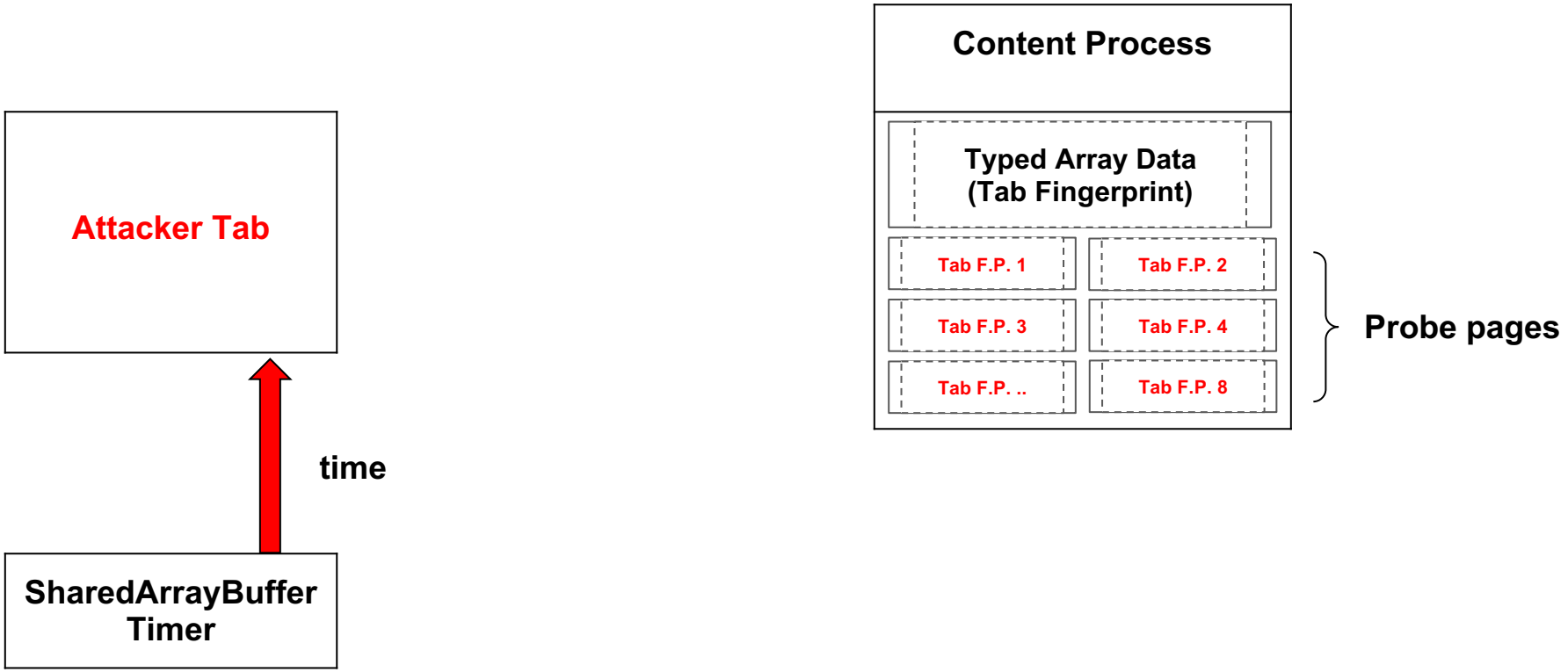
Browser cross-tab scenario: Probing

Attacker Tab

SharedArrayBuffer
Timer



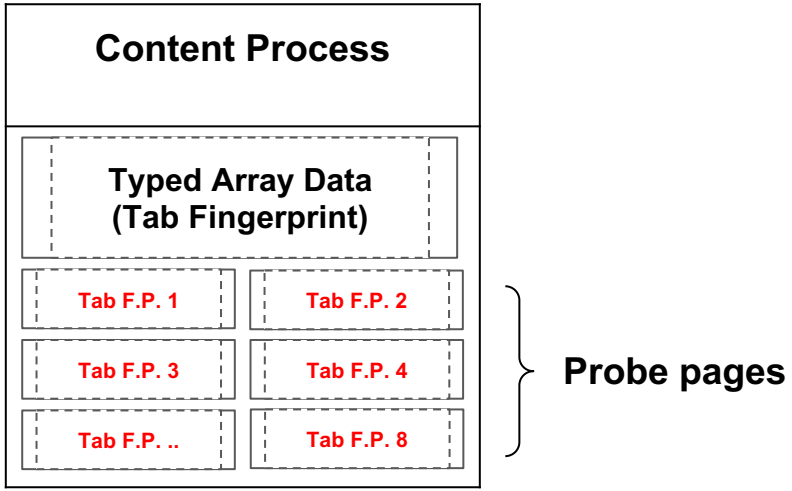
Browser cross-tab scenario: Probing



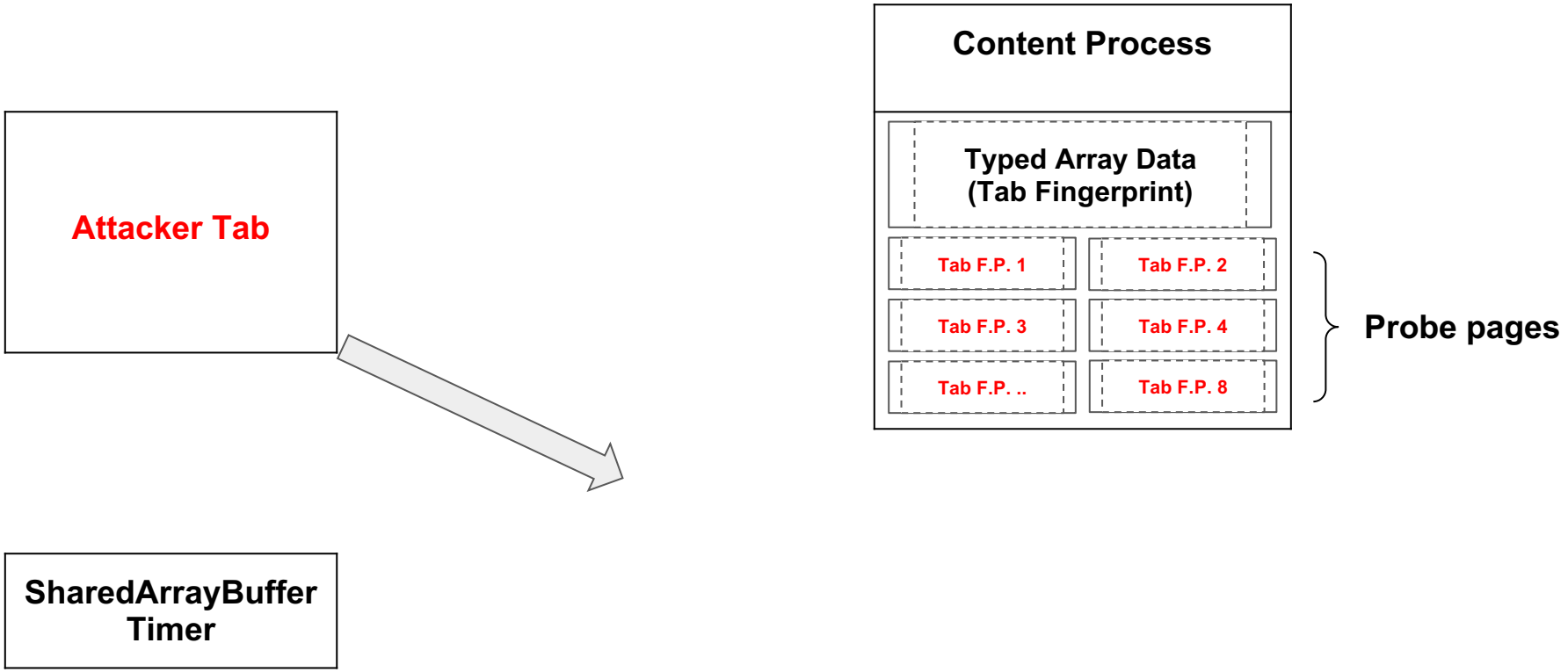
Browser cross-tab scenario: Probing

Attacker Tab

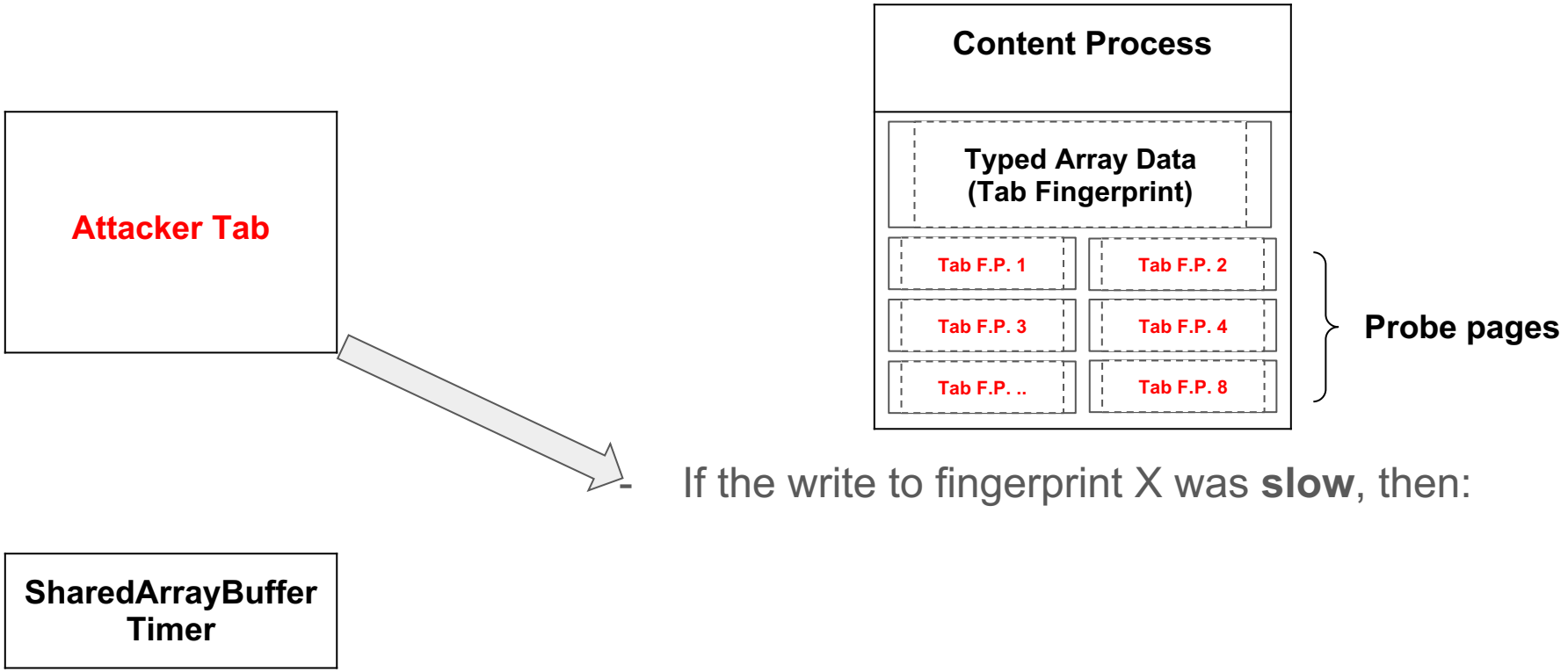
SharedArrayBuffer
Timer



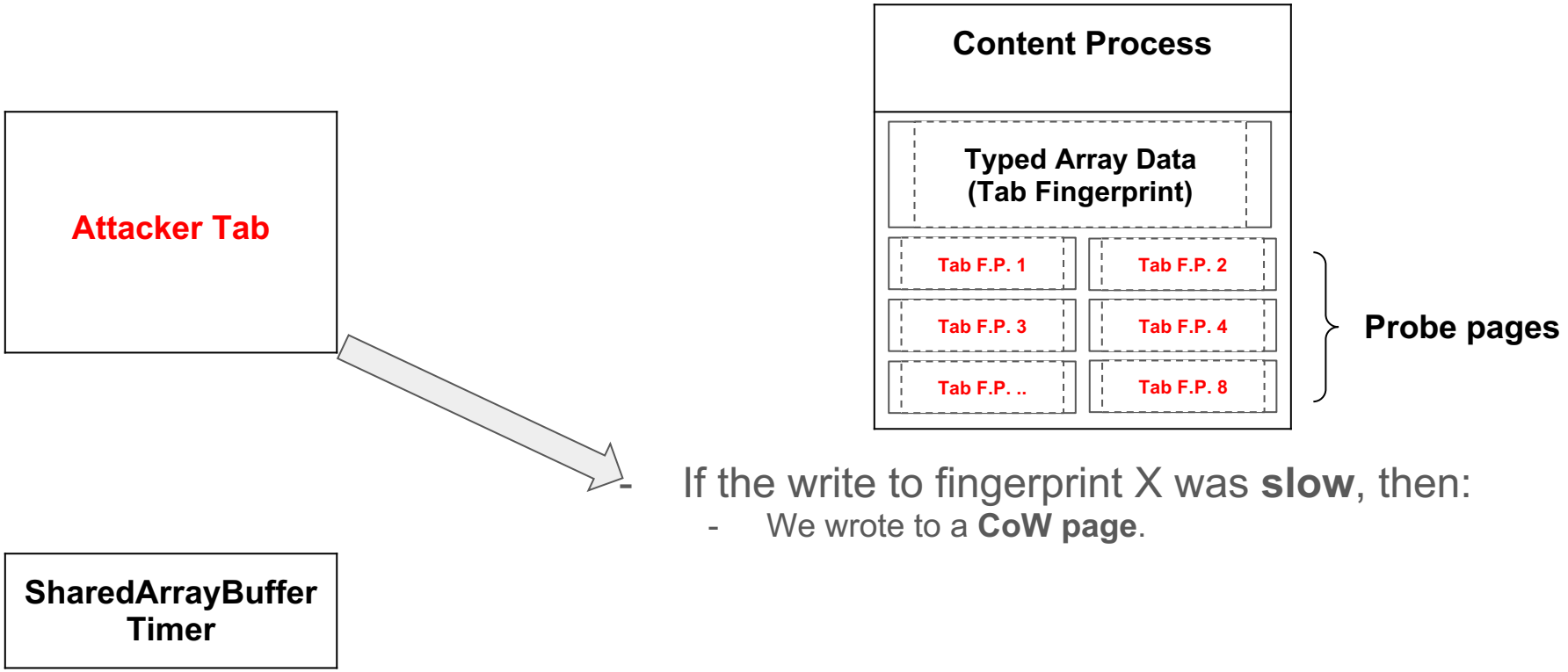
Browser cross-tab scenario: Probing



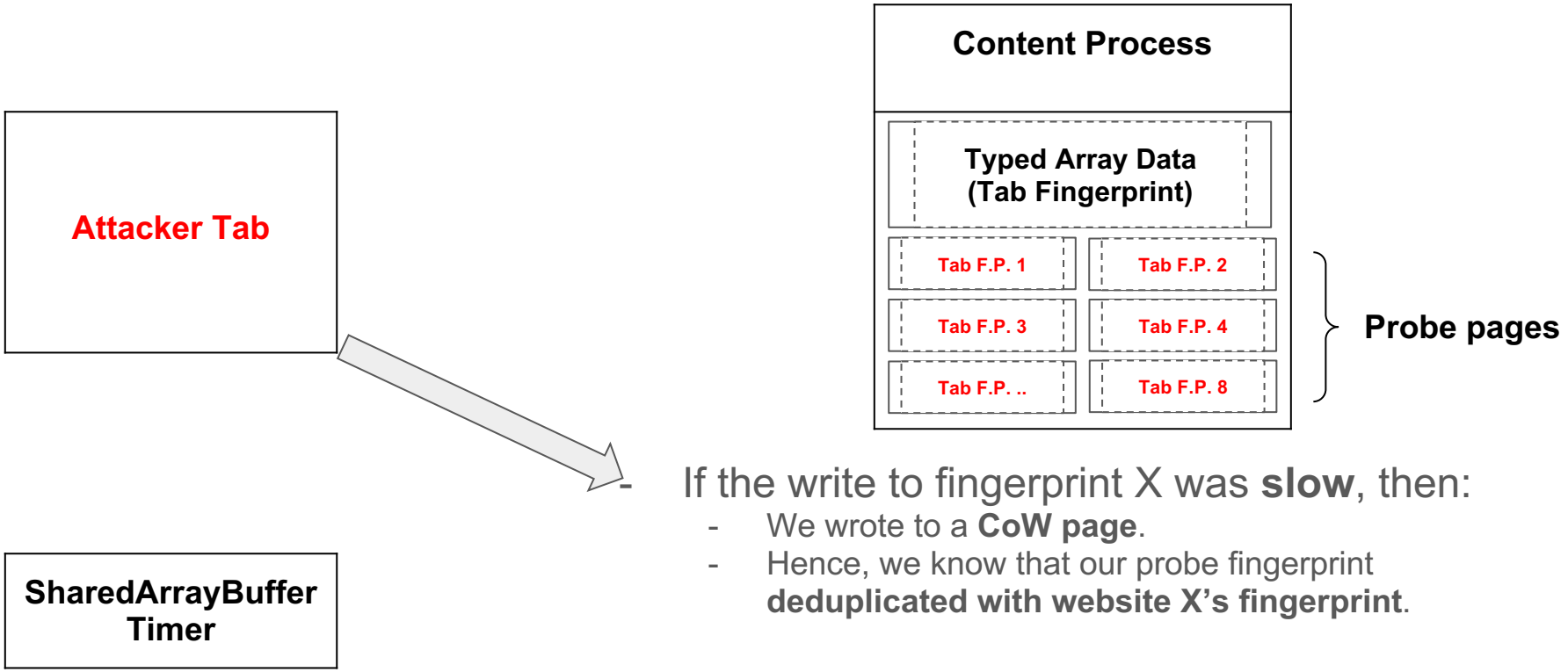
Browser cross-tab scenario: Probing



Browser cross-tab scenario: Probing



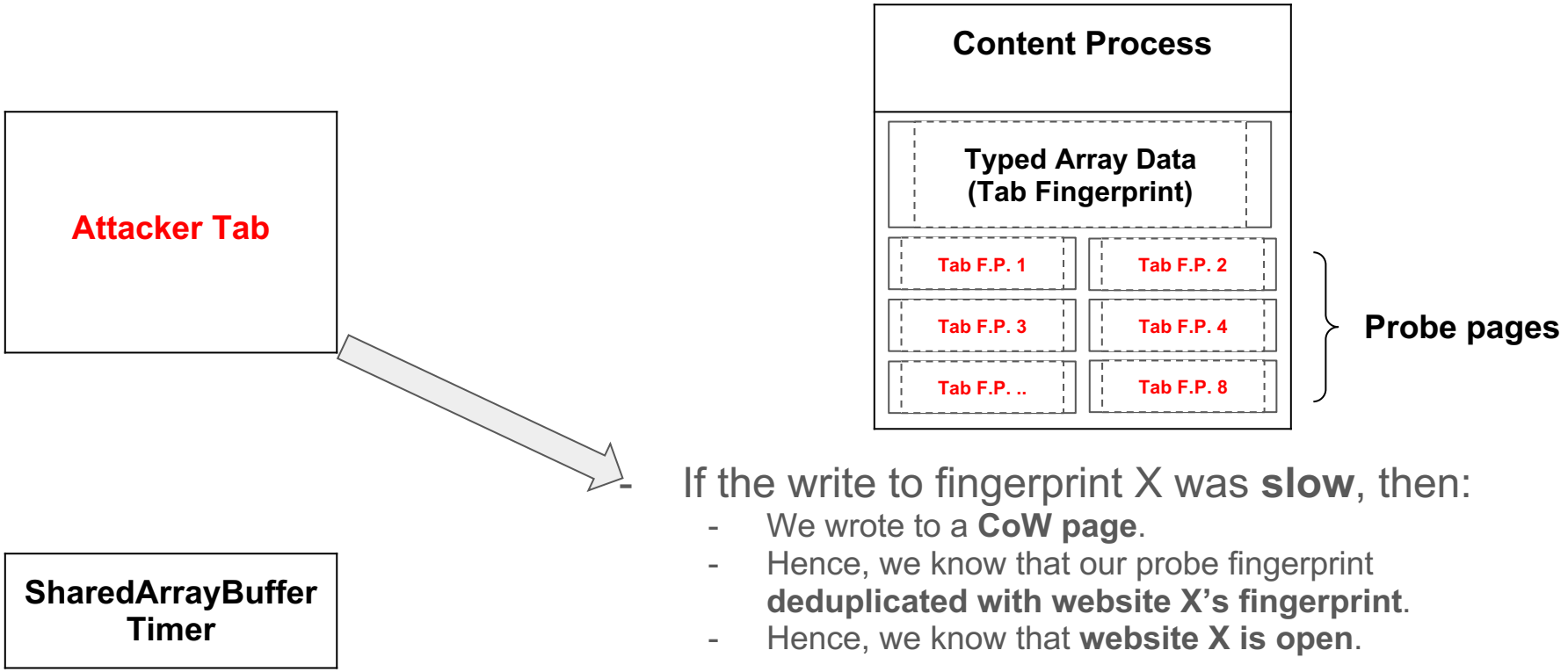
Browser cross-tab scenario: Probing



If the write to fingerprint X was **slow**, then:

- We wrote to a **CoW page**.
- Hence, we know that our probe fingerprint **deduplicated with website X's fingerprint**.

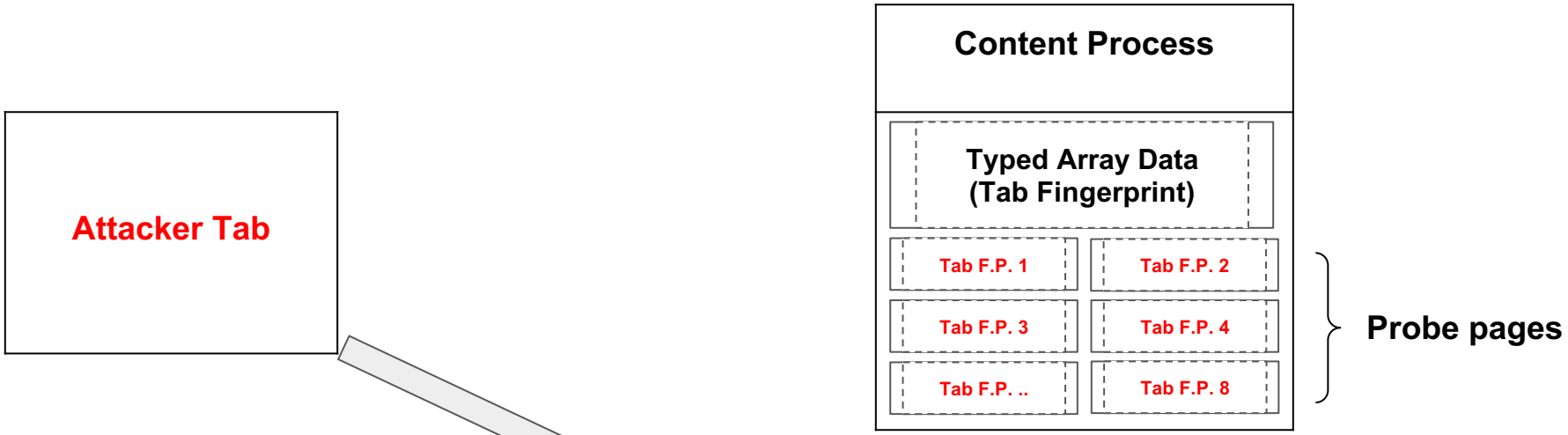
Browser cross-tab scenario: Probing



If the write to fingerprint X was **slow**, then:

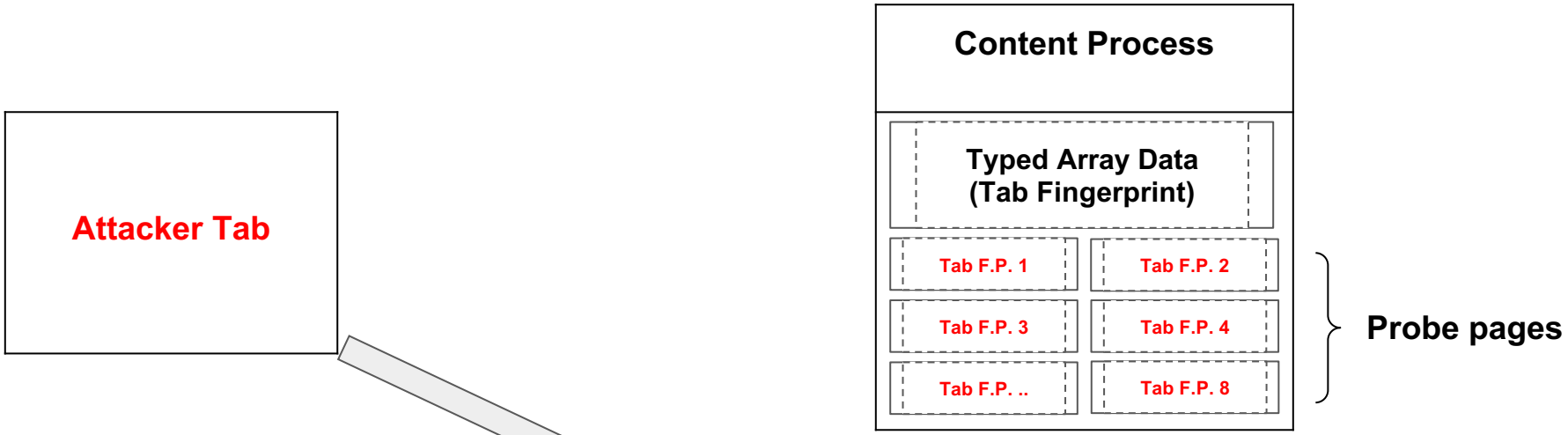
- We wrote to a **CoW page**.
- Hence, we know that our probe fingerprint **deduplicated with website X's fingerprint**.
- Hence, we know that **website X is open**.

Browser cross-tab scenario: Probing



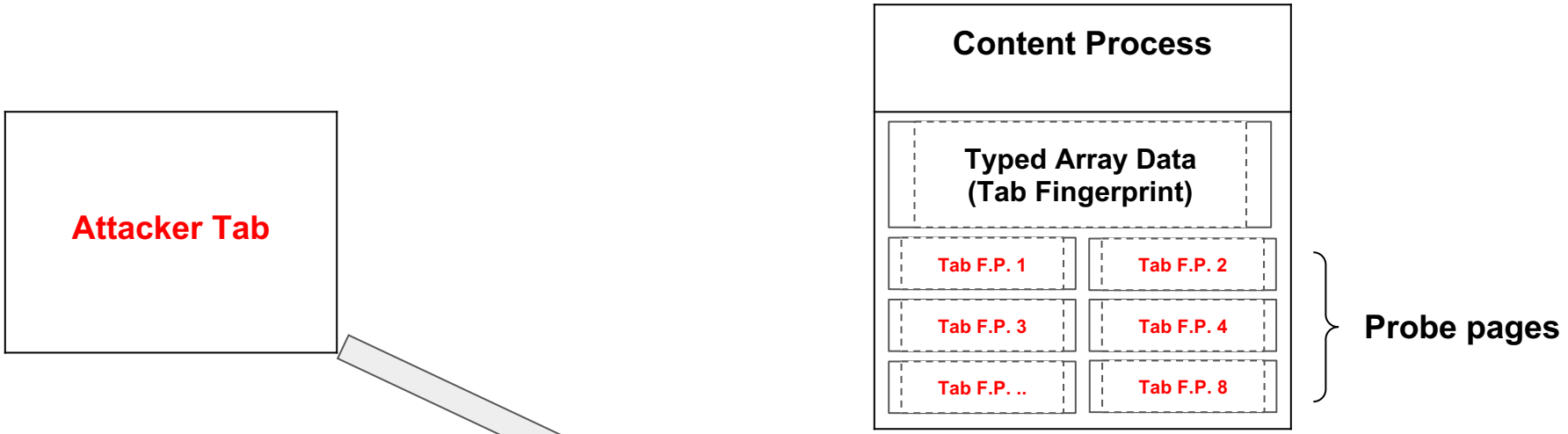
- If the write to fingerprint X was **slow**, then:
  - We wrote to a **CoW page**.
  - Hence, we know that our probe fingerprint **deduplicated with website X's fingerprint**.
  - Hence, we know that **website X is open**.
- If the write to fingerprint Y was **fast**, then:

Browser cross-tab scenario: Probing



- If the write to fingerprint X was **slow**, then:
  - We wrote to a **CoW page**.
  - Hence, we know that our probe fingerprint **deduplicated with website X's fingerprint**.
  - Hence, we know that **website X is open**.
- If the write to fingerprint Y was **fast**, then:
  - We wrote to a **normal page**.

Browser cross-tab scenario: Probing



- If the write to fingerprint X was **slow**, then:
  - We wrote to a **CoW page**.
  - Hence, we know that our probe fingerprint **deduplicated with website X's fingerprint**.
  - Hence, we know that **website X is open**.
- If the write to fingerprint Y was **fast**, then:
  - We wrote to a **normal page**.
  - We can only conclude that we're not sharing a process with website Y.

Browser cross-tab scenario: Conclusion

Browser cross-tab scenario: Conclusion

This scenario highlights **how difficult it is to conform to the model** assumed by **security domain-based deduplication**:

Browser cross-tab scenario: Conclusion

This scenario highlights **how difficult it is to conform to the model** assumed by **security domain-based deduplication**:

- I.e., where **programs are rewritten** such that processes that handle **untrusted data** are **separated from** processes that handle **trusted data**.

Browser cross-tab scenario: Conclusion

This scenario highlights **how difficult it is to conform to the model** assumed by **security domain-based deduplication**:

- I.e., where **programs are rewritten** such that processes that handle **untrusted data** are **separated from** processes that handle **trusted data**.
- Such a code rewrite is **non-trivial** in practice.

Browser cross-tab scenario: Conclusion

This scenario highlights **how difficult it is to conform to the model** assumed by **security domain-based deduplication**:

- I.e., where **programs are rewritten** such that processes that handle **untrusted data** are **separated from** processes that handle **trusted data**.
- Such a code rewrite is **non-trivial** in practice.
- In particular, Firefox didn’t **mitigate** this until **November 2021**, when it adopted full site isolation.

Conclusion

Conclusion

- Deduplication-based side channel attacks are still feasible because it is **non-trivial to separate programs into separate security domains**.

Conclusion

- Deduplication-based side channel attacks are still feasible because it is **non-trivial to separate programs into separate security domains**.
- However, not all hope is lost! Promising mitigations exist, e.g.:

Conclusion

- Deduplication-based side channel attacks are still feasible because it is **non-trivial to separate programs into separate security domains**.
- However, not all hope is lost! Promising mitigations exist, e.g.:
  - **Opt-in** security domain-based deduplication, i.e., deduplication is **off by default**.

Conclusion

- Deduplication-based side channel attacks are still feasible because it is **non-trivial to separate programs into separate security domains**.
- However, not all hope is lost! Promising mitigations exist, e.g.:
  - **Opt-in** security domain-based deduplication, i.e., deduplication is **off by default**.
  - **VUsion**, a mechanism that ensures the same behavior on all pages of a system.

Thank you!