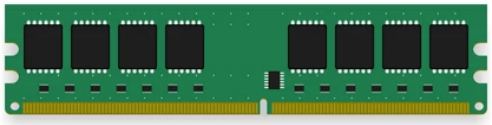
CHERI-picking: Leveraging capability hardware for prefetching

Shaurya Patel, Sid Agrawal, Alexandra (Sasha) Fedorova, Margo Seltzer
University of British Columbia

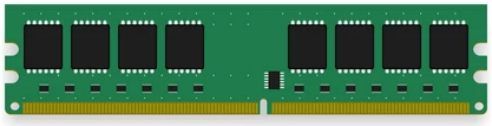
Memory management in the datacenter

Memory management in the datacenter

Memory management in the datacenter

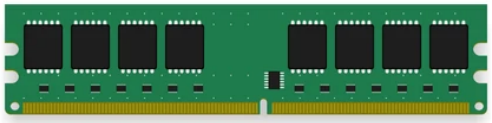


Memory management in the datacenter

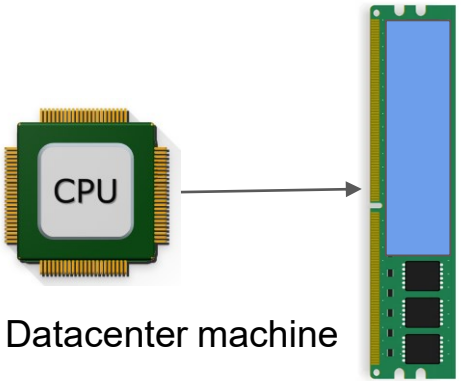


DRAM chips **cost**
30% of a datacenter

Memory management in the datacenter

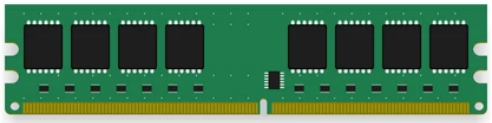


DRAM chips **cost** **30%** of a datacenter

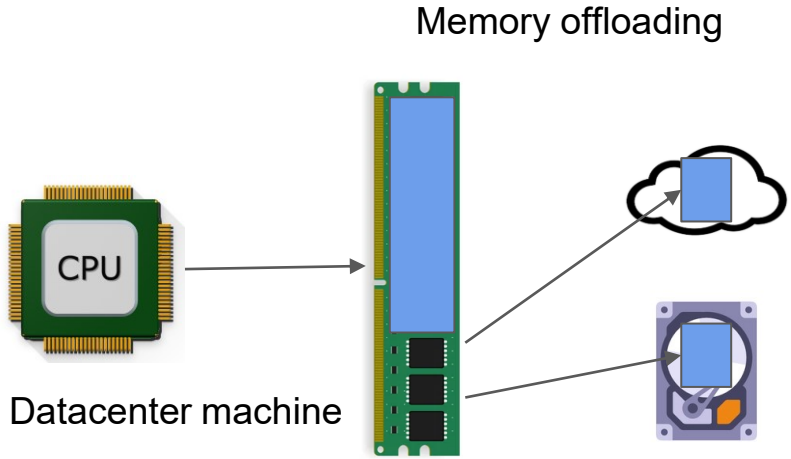


Datacenter machine

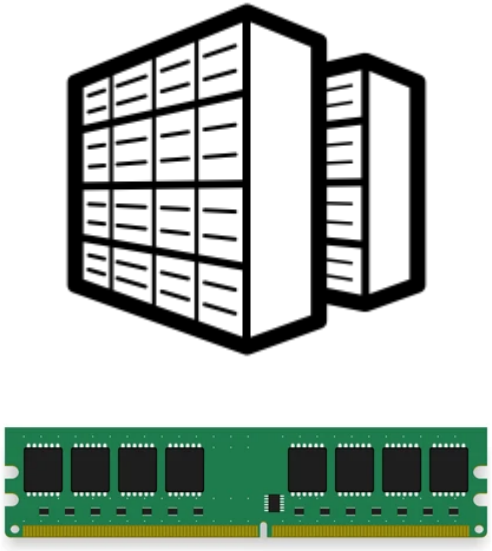
Memory management in the datacenter



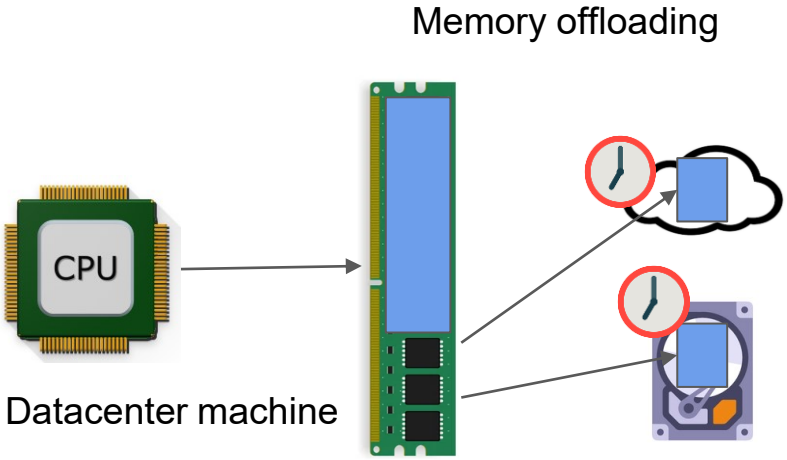
DRAM chips **cost** **30%** of a datacenter



Memory management in the datacenter

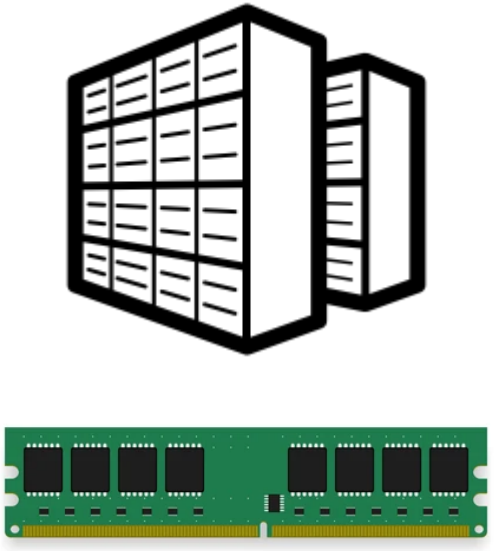


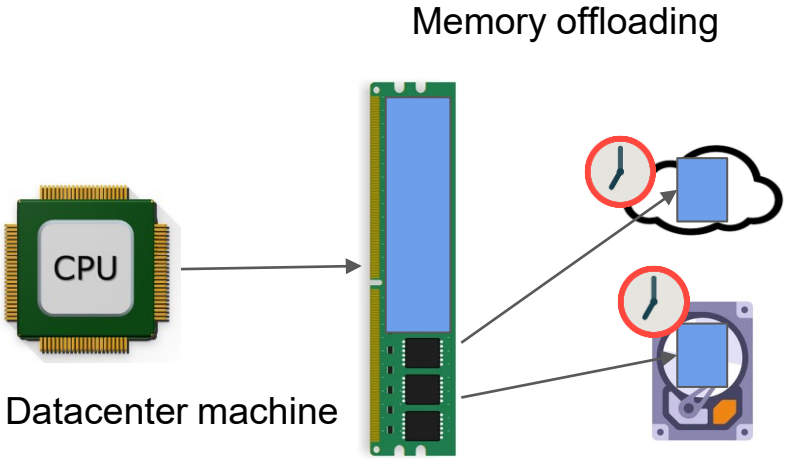
DRAM chips **cost 30%** of a datacenter



Access to these devices is **slower than DRAM**

Memory management in the datacenter





Prefetching memory pages is an **effective** way to minimize overhead

Problem with current kernel prefetchers

Problem with current kernel prefetchers

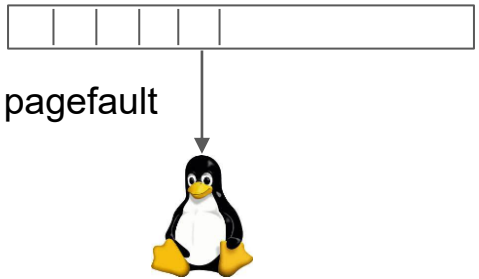
Sequential patterns:

```
for( i = 0; i < 1000; i++) {
        b = a[i];
        …
}
```

Problem with current kernel prefetchers

Sequential patterns:

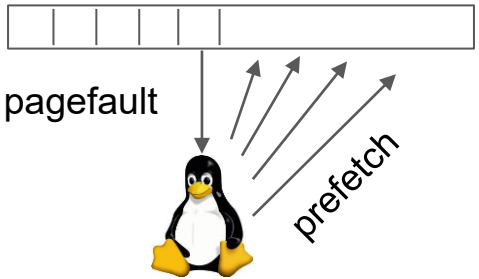
```
for( i = 0; i < 1000; i++) {
        b = a[i];
        …
}
```



Problem with current kernel prefetchers

Sequential patterns:

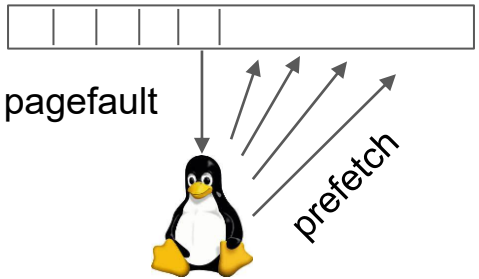
```
for( i = 0; i < 1000; i++) {
        b = a[i];
        …
}
```



Problem with current kernel prefetchers

Sequential patterns:

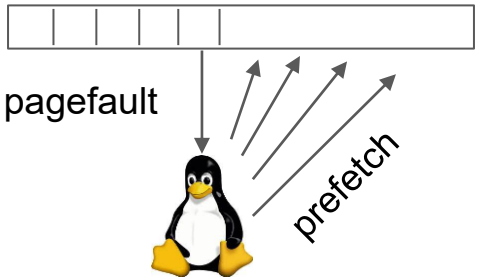
```
for( i = 0; i < 1000; i++) {
        b = a[i];
        …
}
```



Problem with current kernel prefetchers

Sequential patterns:

```
for( i = 0; i < 1000; i++) {
        b = a[i];
        …
}
```

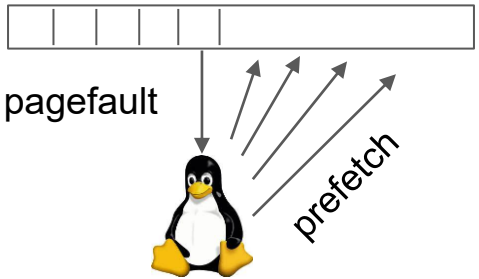


Current kernel prefetchers are **good with sequential** accesses that show regular access patterns

Problem with current kernel prefetchers

Sequential patterns:

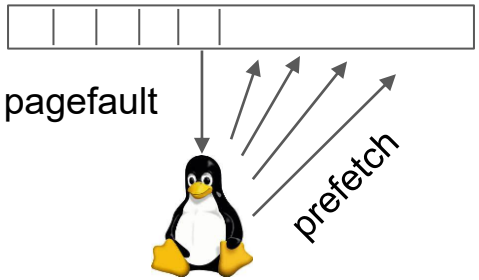
```
for( i = 0; i < 1000; i++) {
        b = a[i];
        …
}
```



Problem with current kernel prefetchers

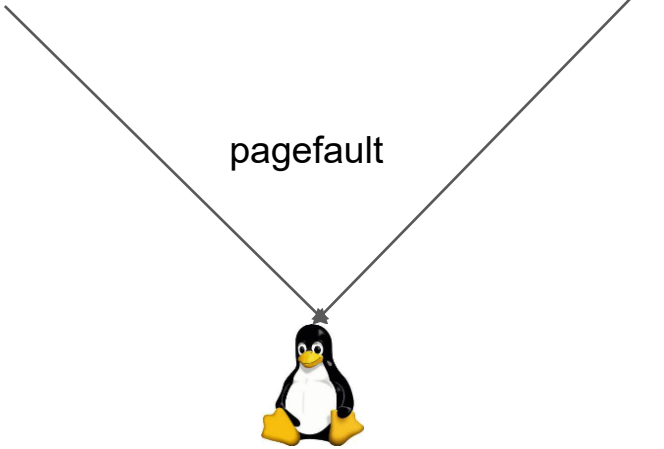
Sequential patterns:

```
for( i = 0; i < 1000; i++) {
        b = a[i];
        …
}
```




```
User u = session.getUser();
Account a = u.getAccount();
Balance b = a.getBalance();
```

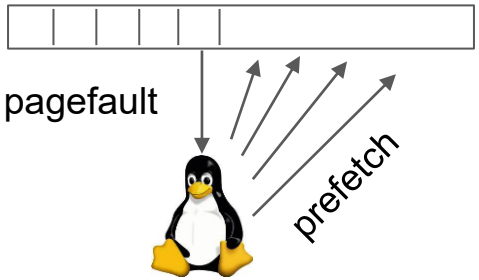
```
while(!curr) {
        curr = curr->next
}
```



Problem with current kernel prefetchers

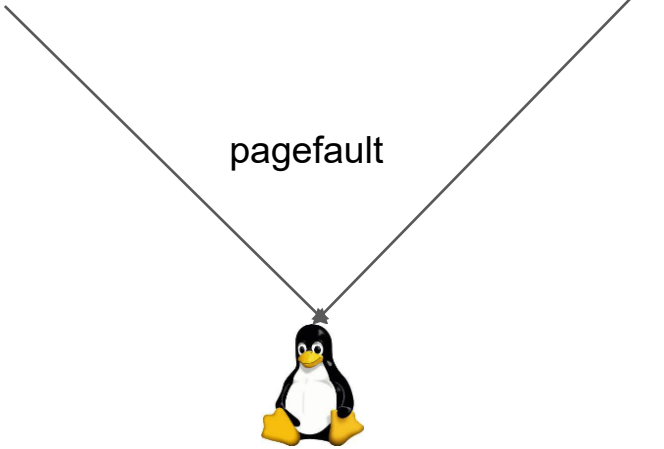
Sequential patterns:

```
for( i = 0; i < 1000; i++) {
        b = a[i];
        …
}
```




```
User u = session.getUser();
Account a = u.getAccount();
Balance b = a.getBalance();
```

```
while(!curr) {
        curr = curr->next
}
```



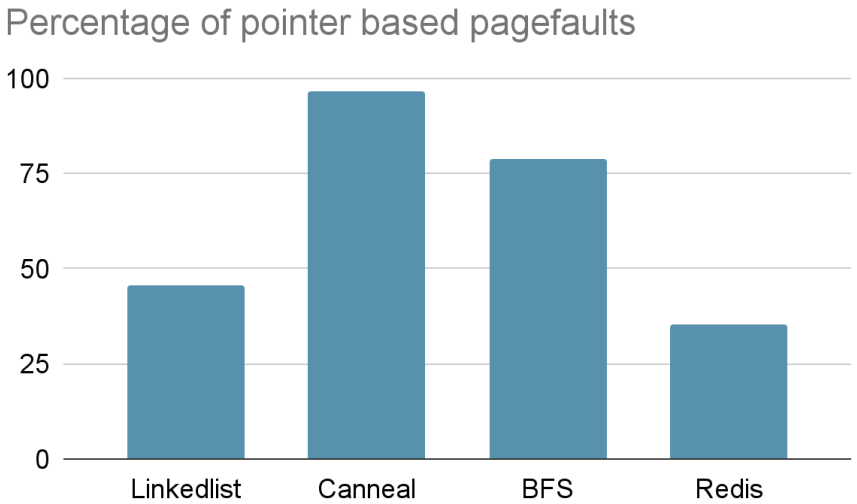
Current kernel prefetchers are **ineffective for irregular patterns** such as reference or pointer based patterns

Do pointer-based patterns even exist?

- What percentage of pointer accesses cause page faults?
- What is the performance of the current default prefetcher on those pagefaults?

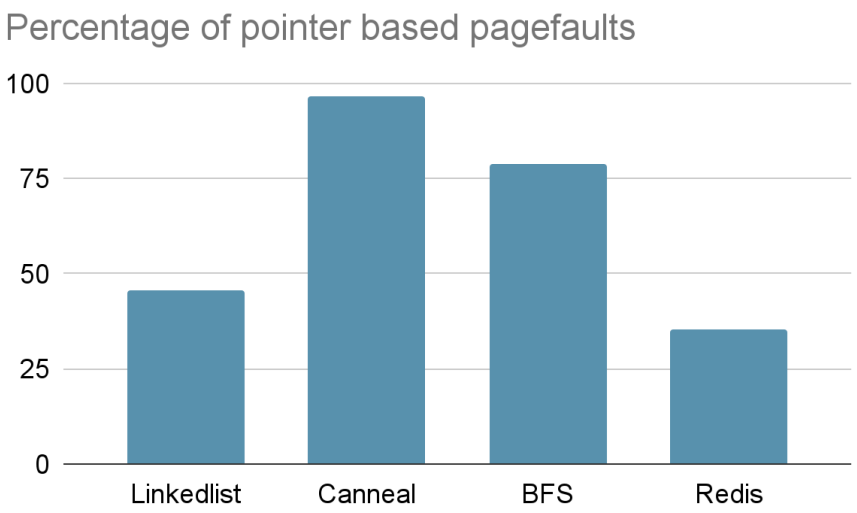
Do pointer-based patterns even exist?

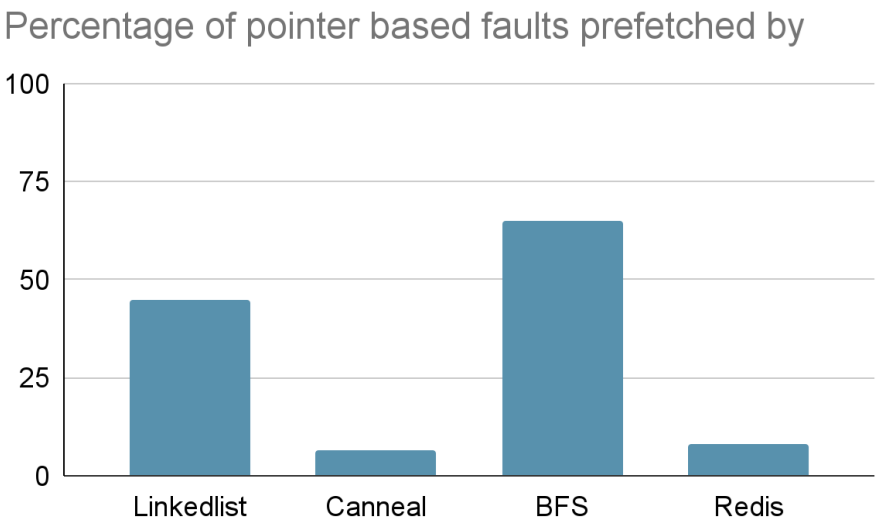
- What percentage of pointer accesses cause page faults?
- What is the performance of the current default prefetcher on those pagefaults?



Do pointer-based patterns even exist?

- What percentage of pointer accesses cause page faults?
- What is the performance of the current default prefetcher on those pagefaults?





CHERI-picking

Coverage: The percentage of page faults that were satisfied by previously prefetched pages.

Approach	Application agnostic	Coverage
Strided kernel prefetcher		
Application specific approaches [1][2]		
CHERI-picking		

[1] Canvas: Isolated and Adaptive Swapping for Multi-Applications on Remote Memory. Wang et. al. NSDI 2023
[2] DiLOS: Do Not Trade Compatibility for Performance in Memory Disaggregation. Yoon et. al. Eurosys 2023

CHERI-picking

Coverage: The percentage of page faults that were satisfied by previously prefetched pages.

Approach	Application agnostic	Coverage
Strided kernel prefetcher		
Application specific approaches [1][2]		
CHERI-picking		

[1] Canvas: Isolated and Adaptive Swapping for Multi-Applications on Remote Memory. Wang et. al. NSDI 2023
[2] DiLOS: Do Not Trade Compatibility for Performance in Memory Disaggregation. Yoon et. al. Eurosys 2023

CHERI-picking

Coverage: The percentage of page faults that were satisfied by previously prefetched pages.

Approach	Application agnostic	Coverage
Strided kernel prefetcher		
Application specific approaches [1][2]		
CHERI-picking		

[1] Canvas: Isolated and Adaptive Swapping for Multi-Applications on Remote Memory. Wang et. al. NSDI 2023
[2] DiLOS: Do Not Trade Compatibility for Performance in Memory Disaggregation. Yoon et. al. Eurosys 2023

CHERI-picking

Coverage: The percentage of page faults that were satisfied by previously prefetched pages.

Approach	Application agnostic	Coverage
Strided kernel prefetcher		
Application specific approaches [1][2]		
CHERI-picking		

[1] Canvas: Isolated and Adaptive Swapping for Multi-Applications on Remote Memory. Wang et. al. NSDI 2023
[2] DiLOS: Do Not Trade Compatibility for Performance in Memory Disaggregation. Yoon et. al. Eurosys 2023

CHERI-picking

Coverage: The percentage of page faults that were satisfied by previously prefetched pages.

Approach	Application agnostic	Coverage
Strided kernel prefetcher		
Application specific approaches [1][2]		
CHERI-picking		

[1] Canvas: Isolated and Adaptive Swapping for Multi-Applications on Remote Memory. Wang et. al. NSDI 2023
[2] DiLOS: Do Not Trade Compatibility for Performance in Memory Disaggregation. Yoon et. al. Eurosys 2023

CHERI-picking

Coverage: The percentage of page faults that were satisfied by previously prefetched pages.

Approach	Application agnostic	Coverage
Strided kernel prefetcher		
Application specific approaches [1][2]		
CHERI-picking		

[1] Canvas: Isolated and Adaptive Swapping for Multi-Applications on Remote Memory. Wang et. al. NSDI 2023
[2] DiLOS: Do Not Trade Compatibility for Performance in Memory Disaggregation. Yoon et. al. Eurosys 2023

CHERI-picking

Coverage: The percentage of page faults that were satisfied by previously prefetched pages.

Approach	Application agnostic	Coverage
Strided kernel prefetcher	✓	✗
Application specific approaches [1][2]	✗	✓
CHERI-picking	✓	✓

[1] Canvas: Isolated and Adaptive Swapping for Multi-Applications on Remote Memory. Wang et. al. NSDI 2023
[2] DiLOS: Do Not Trade Compatibility for Performance in Memory Disaggregation. Yoon et. al. Eurosys 2023

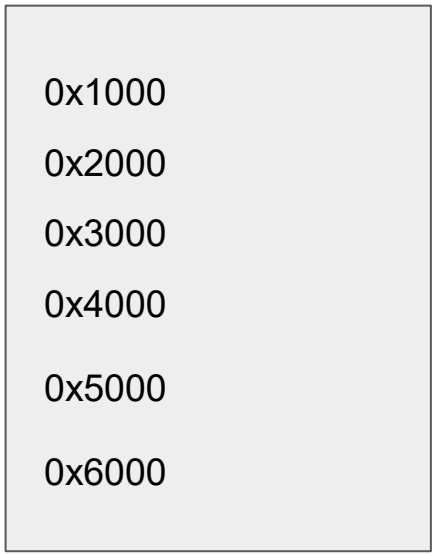
CHERI overview

Capability hardware enhanced RISC instructions (CHERI) treats all pointers as capabilities, and stores a tag bit in hardware for each pointer

CHERI overview

Capability hardware enhanced RISC instructions (CHERI) treats all pointers as capabilities, and stores a tag bit in hardware for each pointer

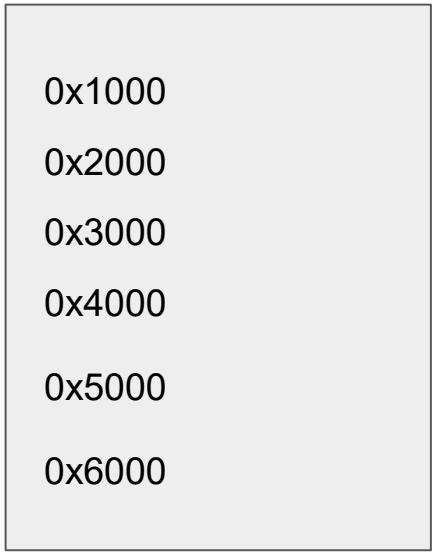
Prior to CHERI



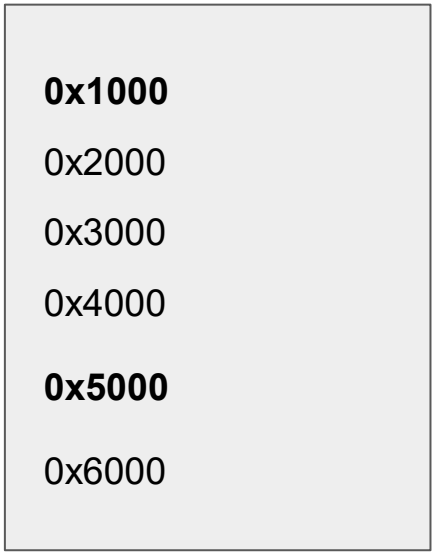
CHERI overview

Capability hardware enhanced RISC instructions (CHERI) treats all pointers as capabilities, and stores a tag bit in hardware for each pointer

Prior to CHERI



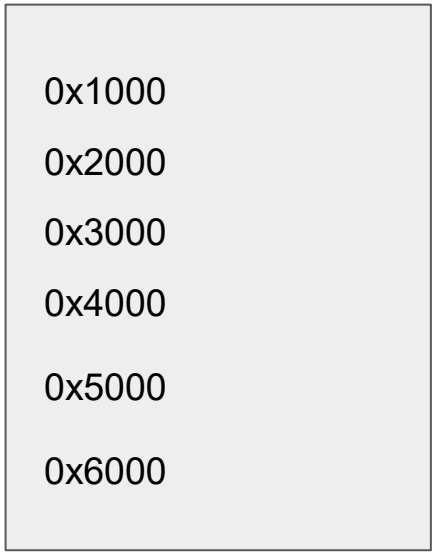
With CHERI



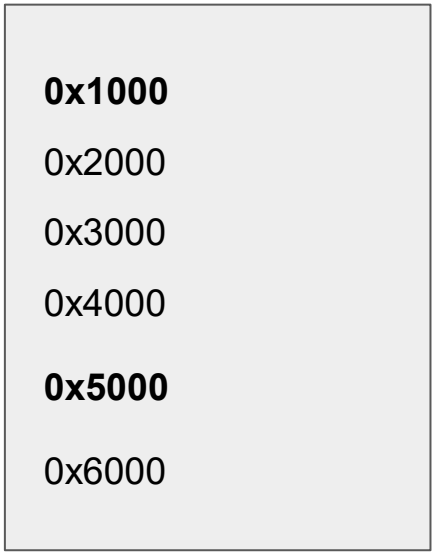
CHERI overview

Capability hardware enhanced RISC instructions (CHERI) treats all pointers as capabilities, and stores a tag bit in hardware for each pointer

Prior to CHERI



With CHERI

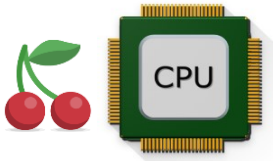


Tag bits
100010

CHERI overview

Capability hardware enhanced RISC instructions (CHERI) treats all pointers as capabilities, and stores a tag bit in hardware for each pointer

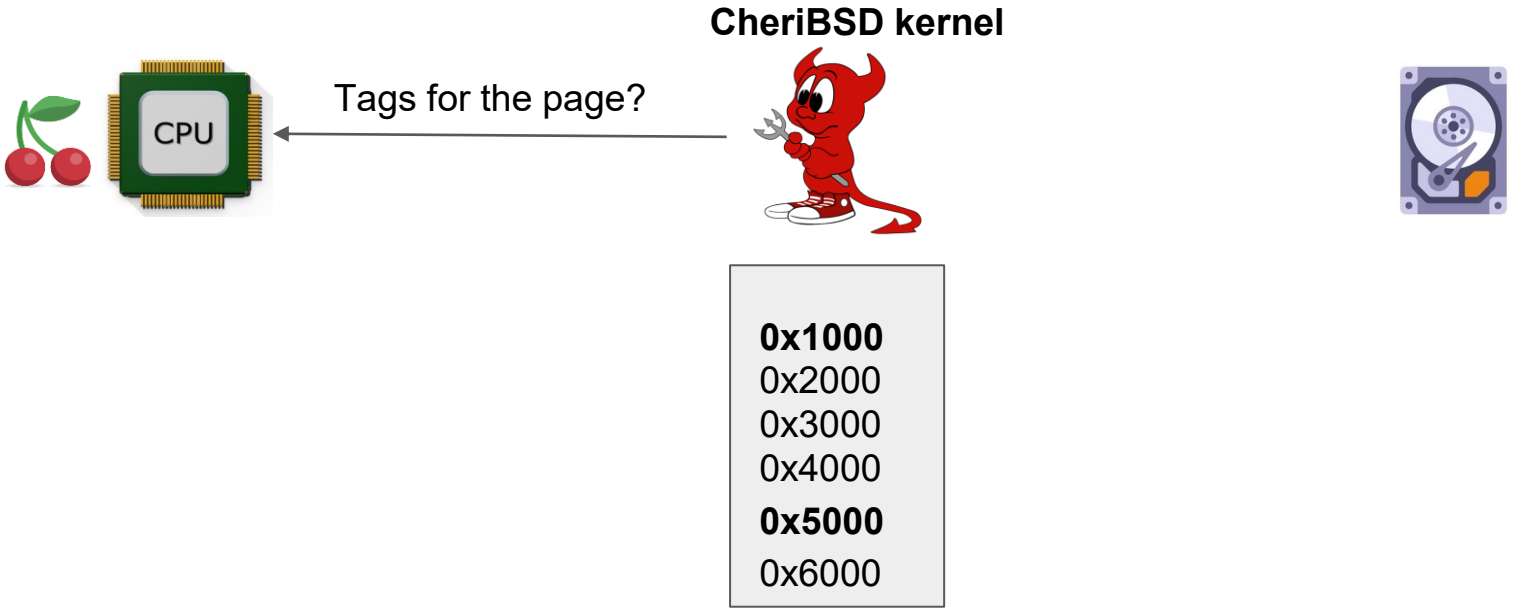
CHERI and swap



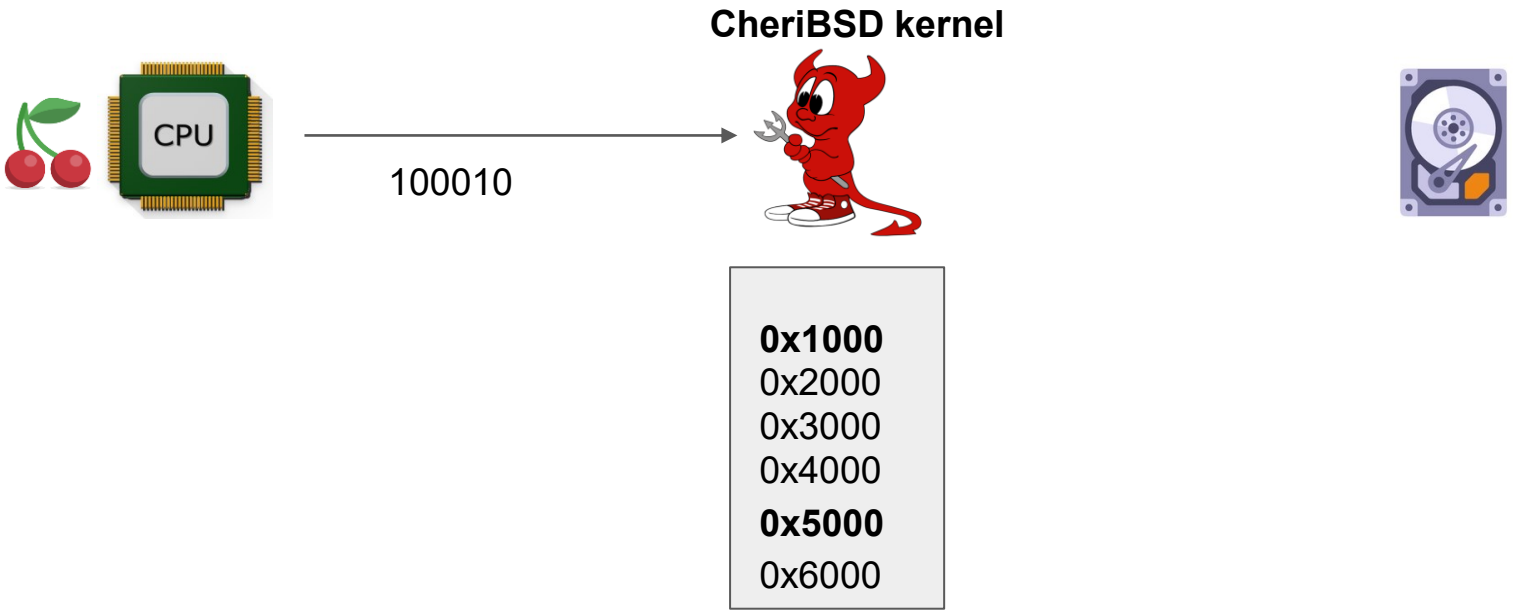
CheriBSD kernel

0x1000
0x2000
0x3000
0x4000
0x5000
0x6000

CHERI and swap

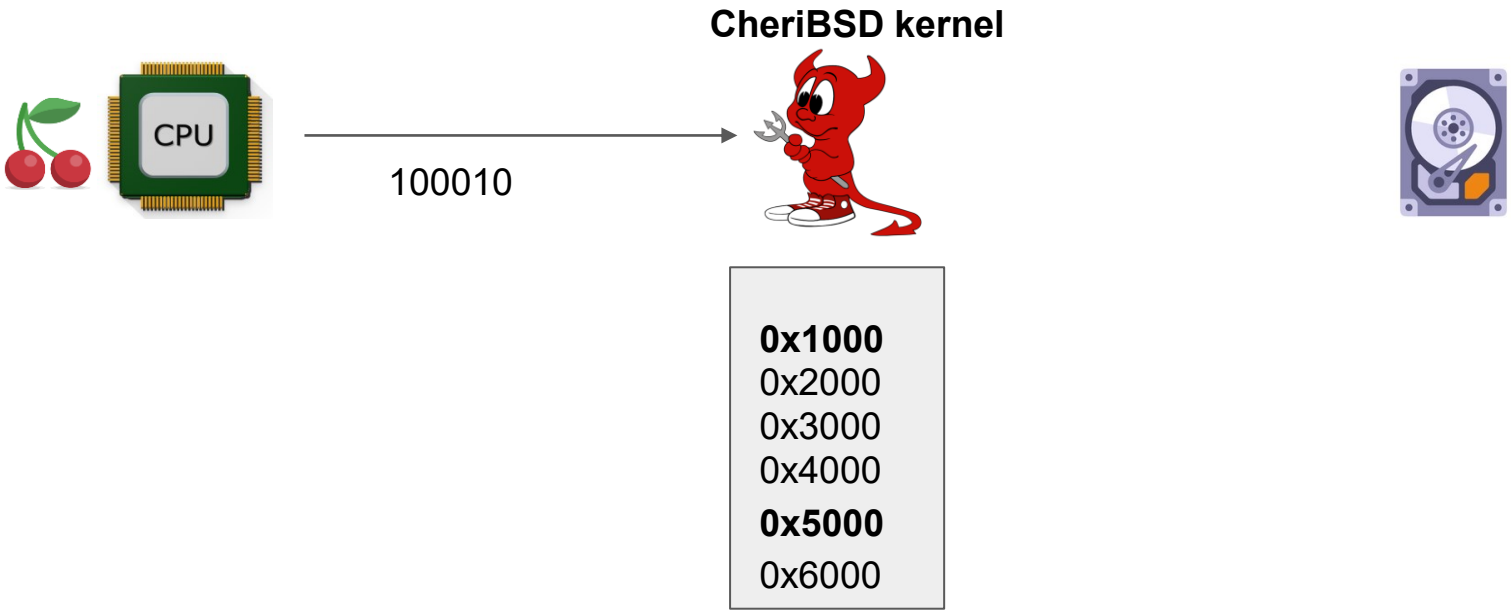


CHERI and swap



CHERI and swap

Internal tag bitmap

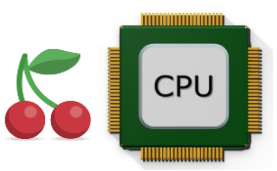


CHERI and swap

Internal tag bitmap

100010

CheriBSD kernel



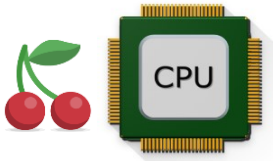
100010

0x1000
0x2000
0x3000
0x4000
0x5000
0x6000

CHERI and swap

Internal tag bitmap

100010



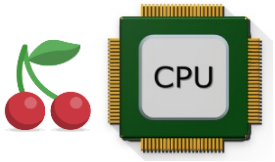
CheriBSD kernel

0x1000
0x2000
0x3000
0x4000
0x5000
0x6000

CHERI and swap

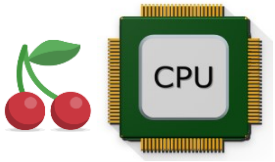
Internal tag bitmap

100010



CheriBSD kernel

CHERI and swap



Internal tag bitmap

100010

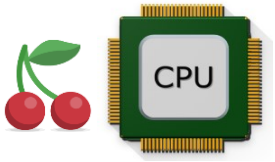
CheriBSD kernel

Move page to swap

CHERI and swap

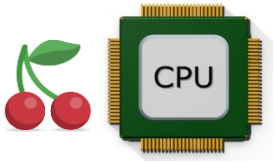
Internal tag bitmap

100010



CheriBSD kernel

CHERI and swap

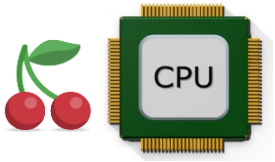


Internal tag bitmap

100010

CheriBSD kernel

CHERI and swap



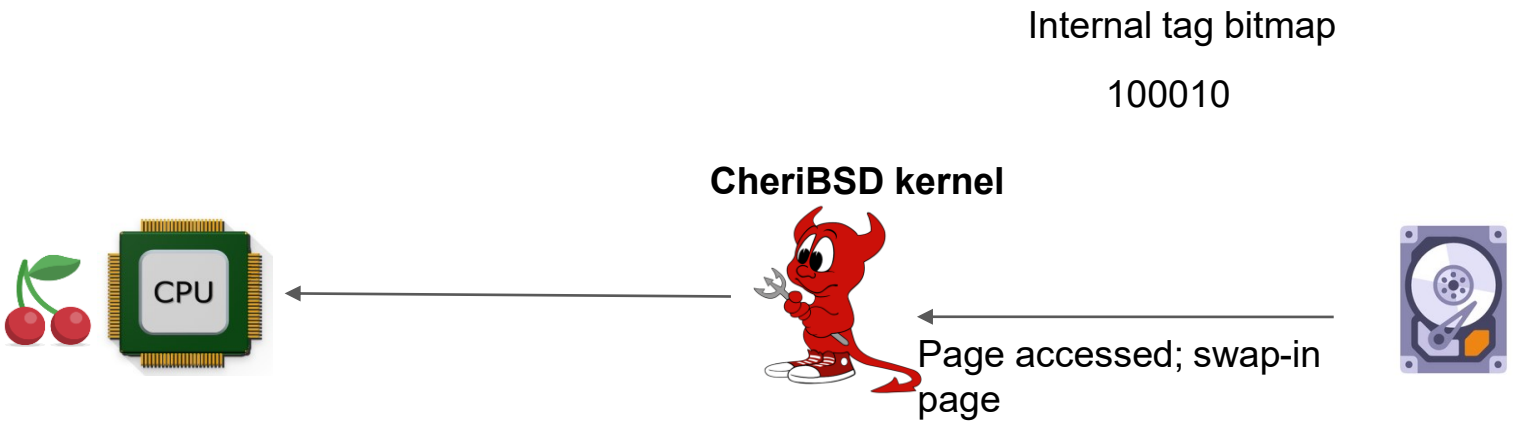
Internal tag bitmap

100010

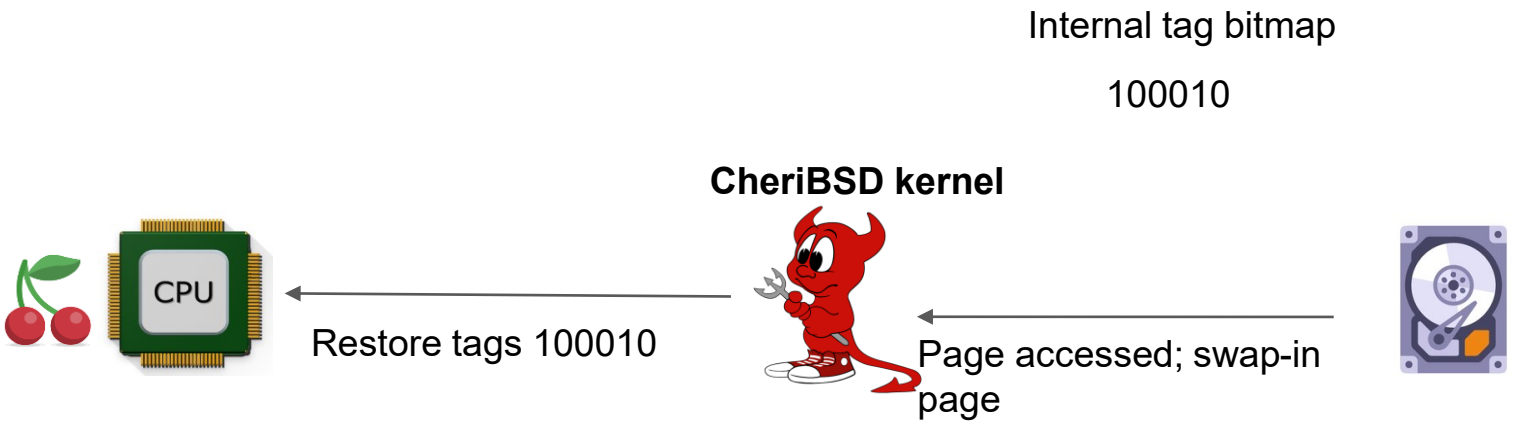
CheriBSD kernel

Page accessed; swap-in page

CHERI and swap



CHERI and swap



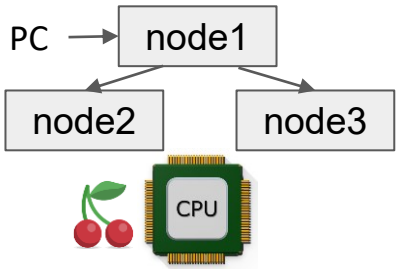
CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.

CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.

Tree traversal



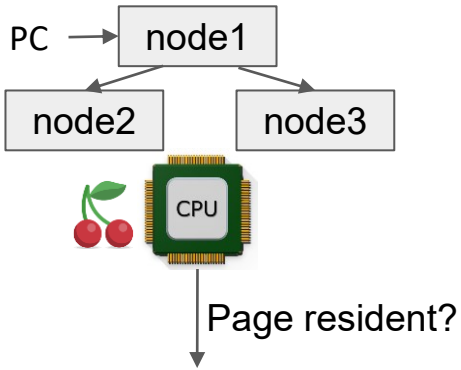
CheriBSD kernel

Node 1	Swap
Node 2	Mapped
Node 3	Swap

CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.

Tree traversal

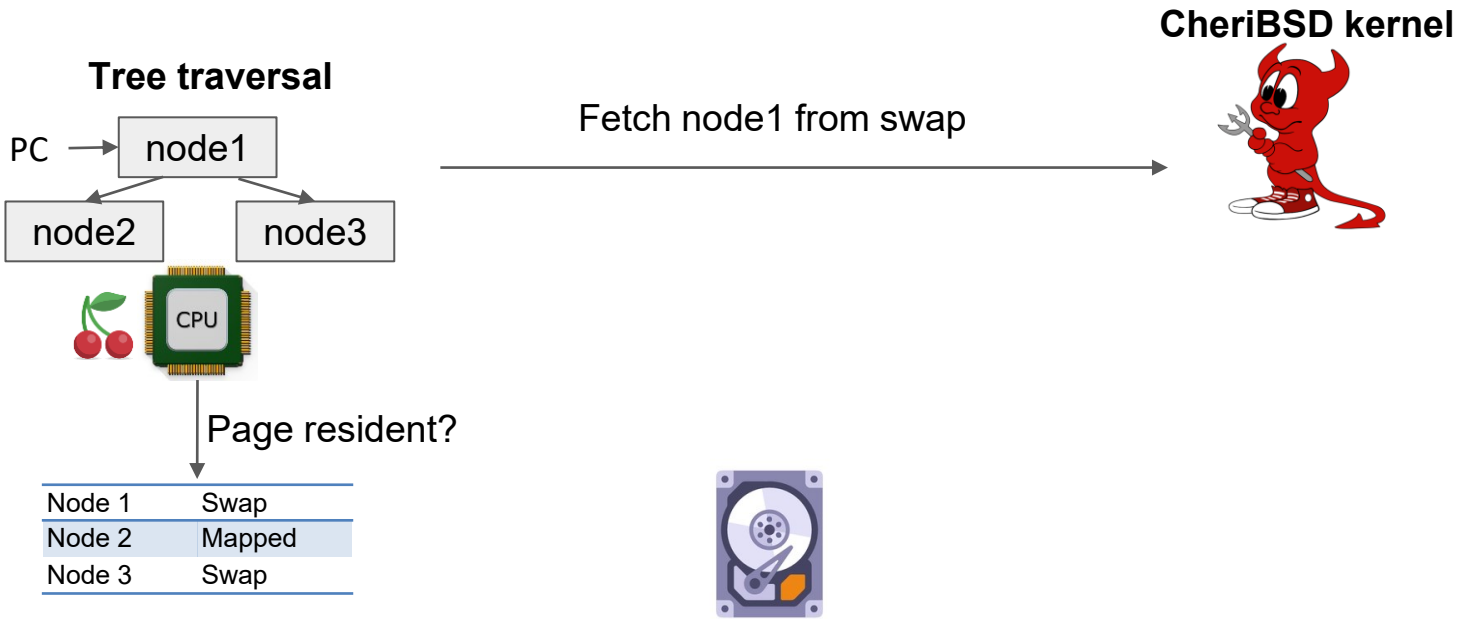


Node 1	Swap
Node 2	Mapped
Node 3	Swap

CheriBSD kernel

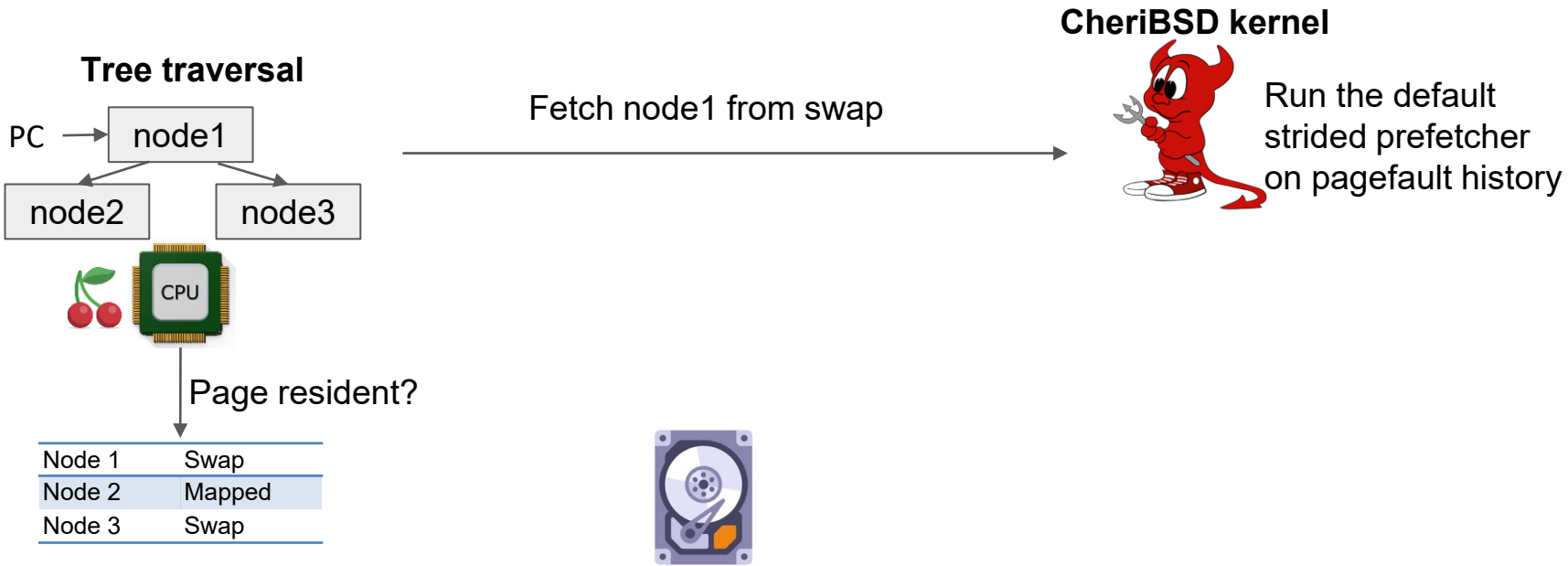
CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.



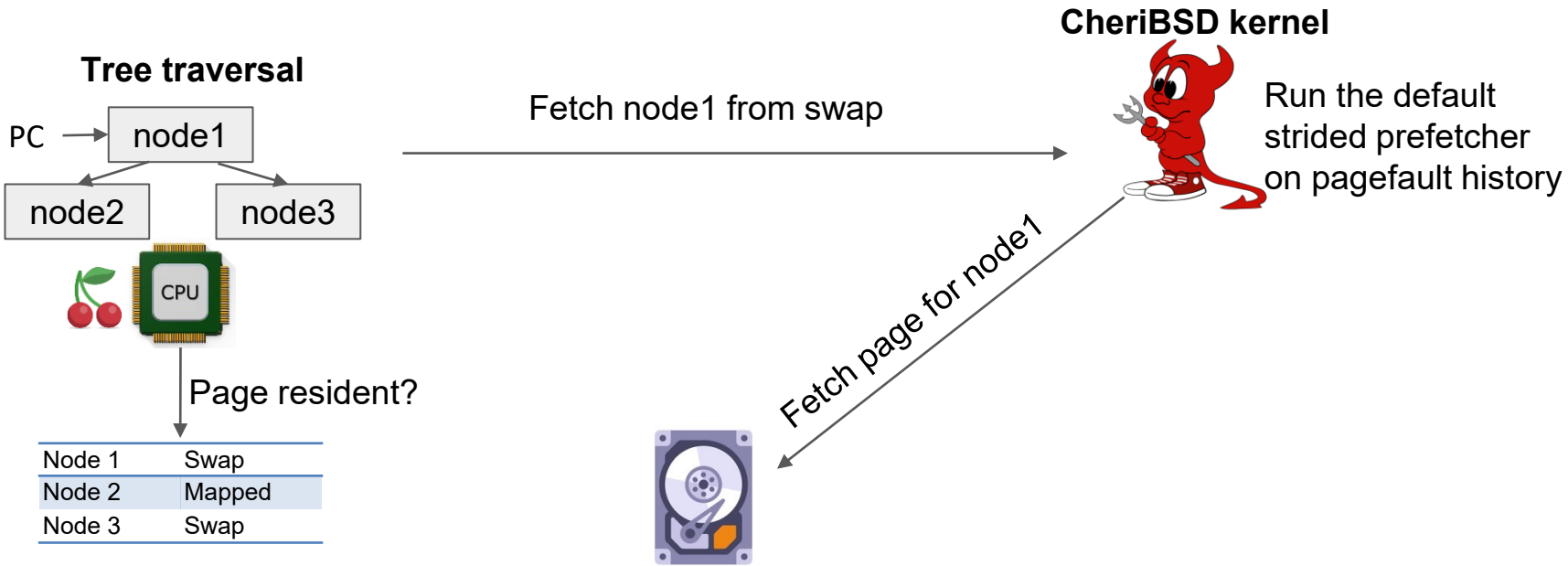
CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.



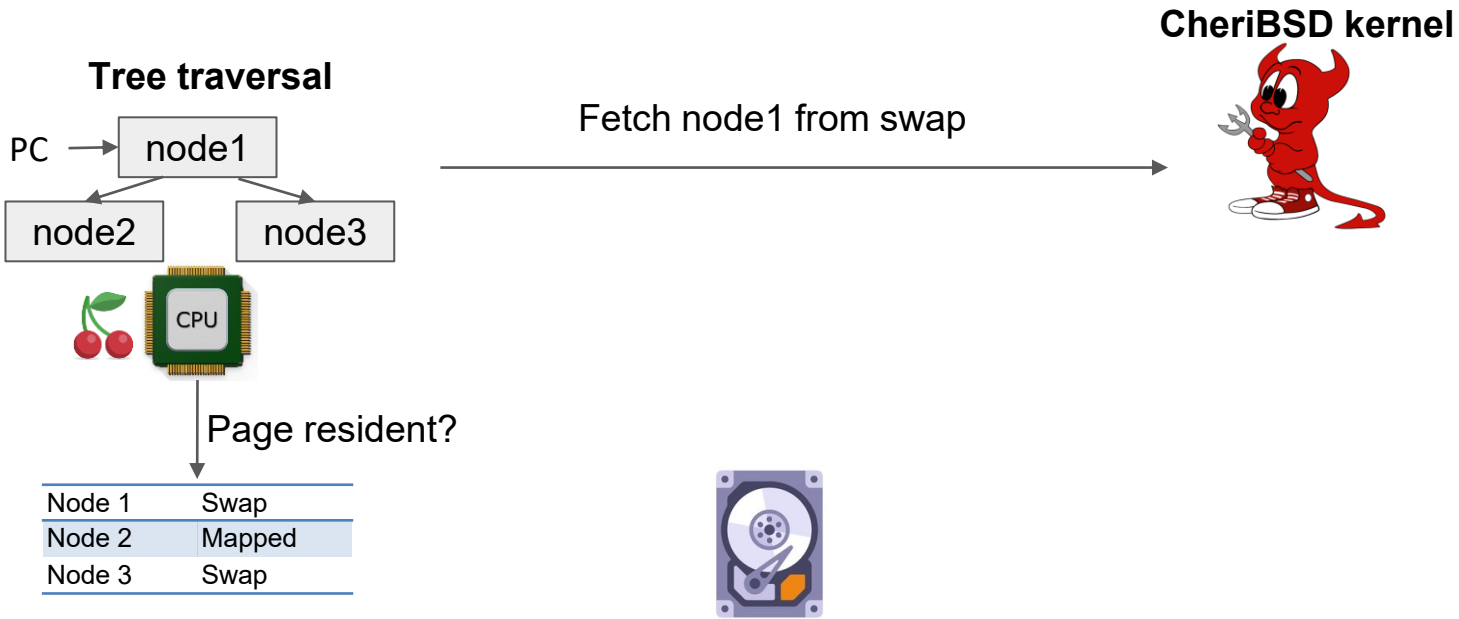
CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.



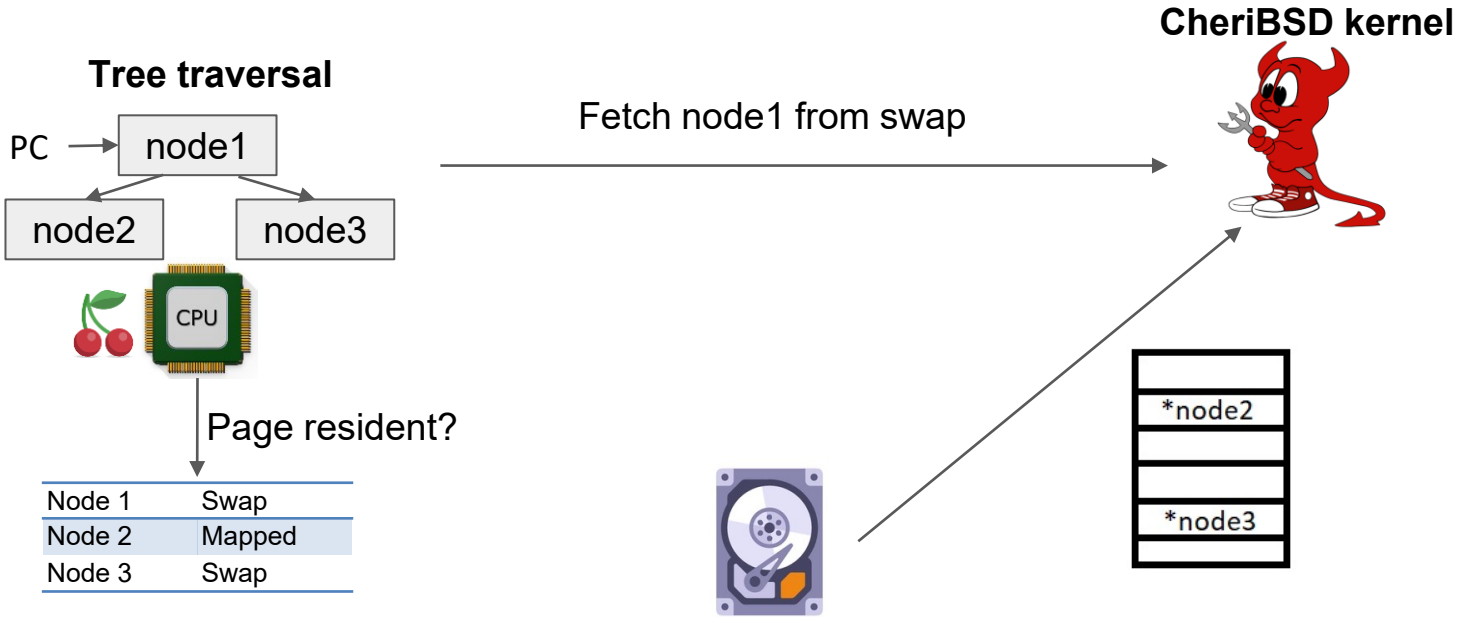
CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.



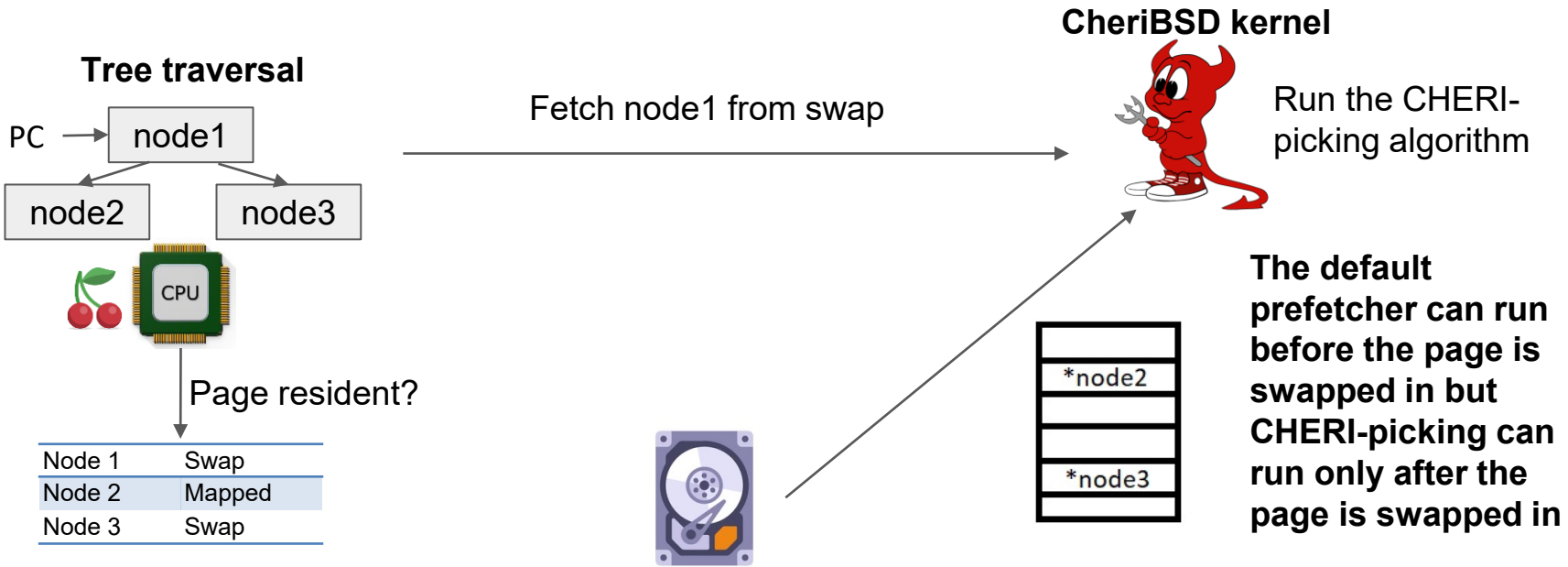
CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.



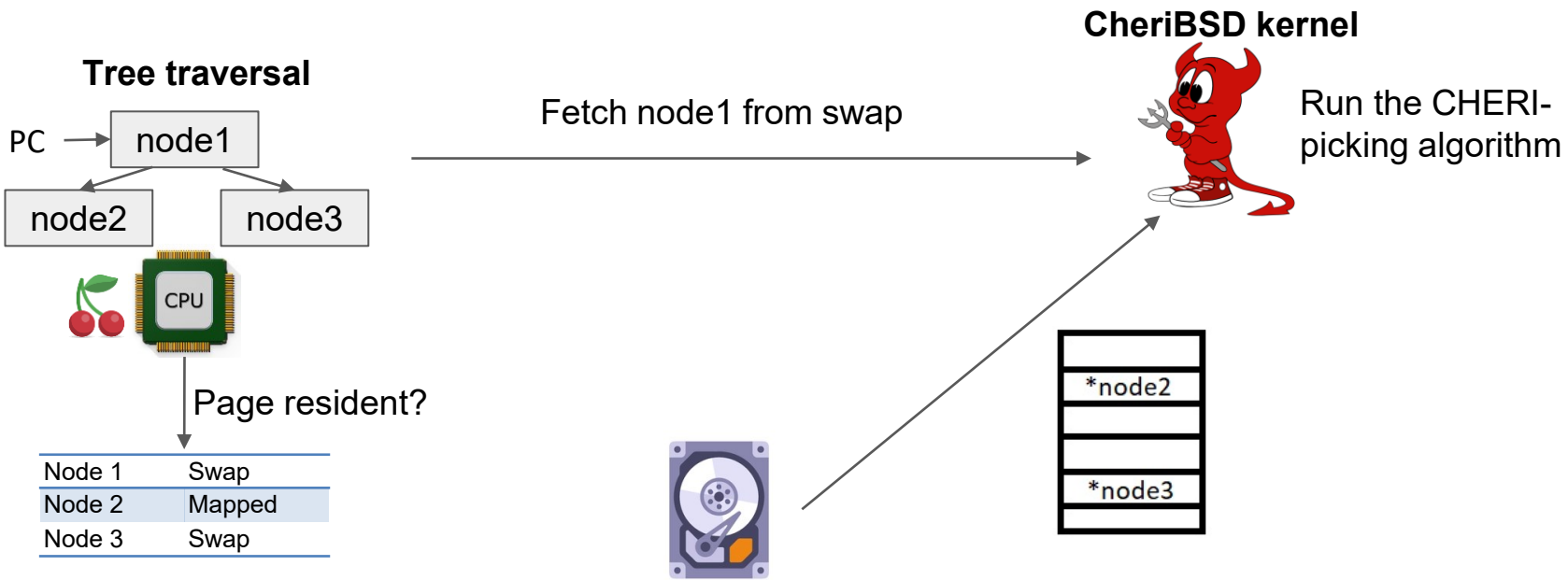
CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.



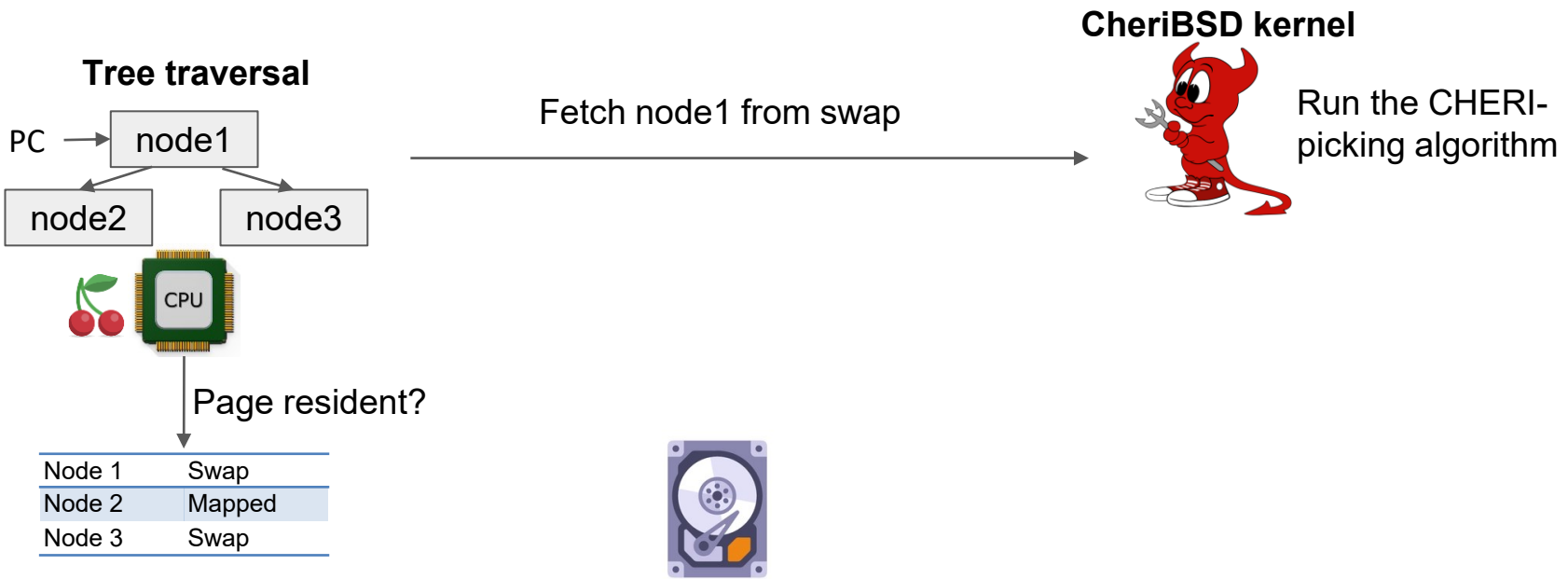
CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.



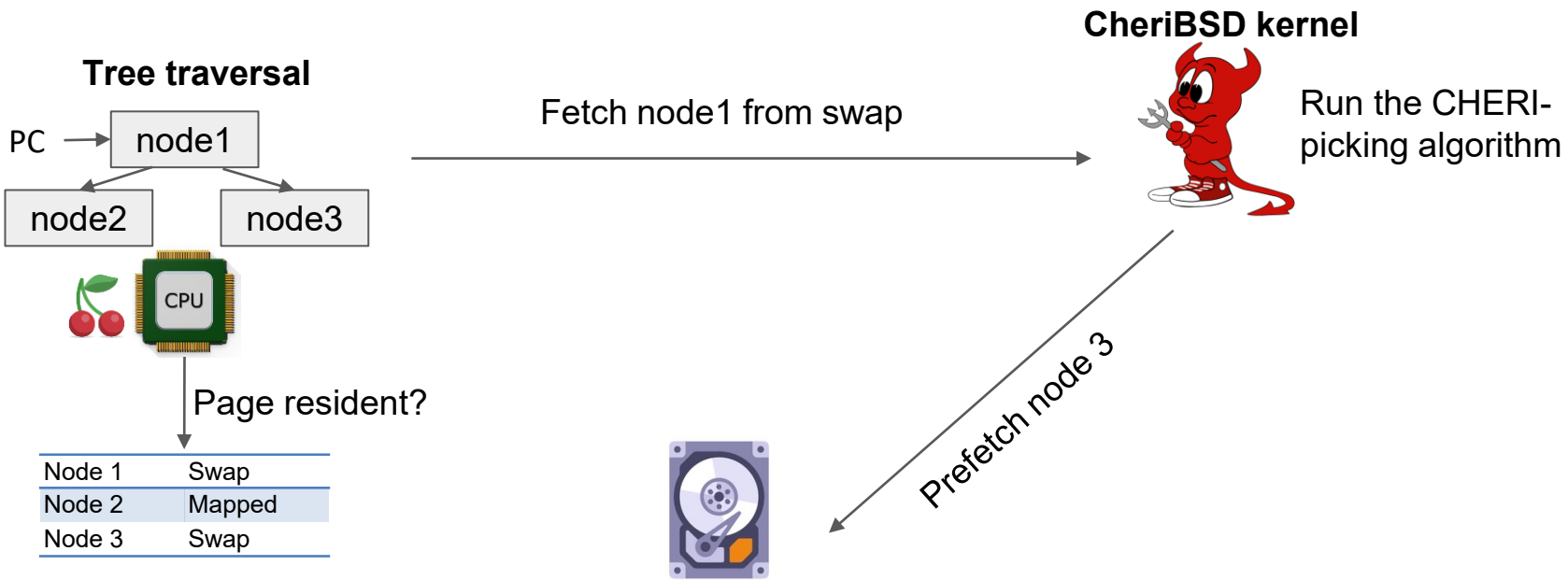
CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.



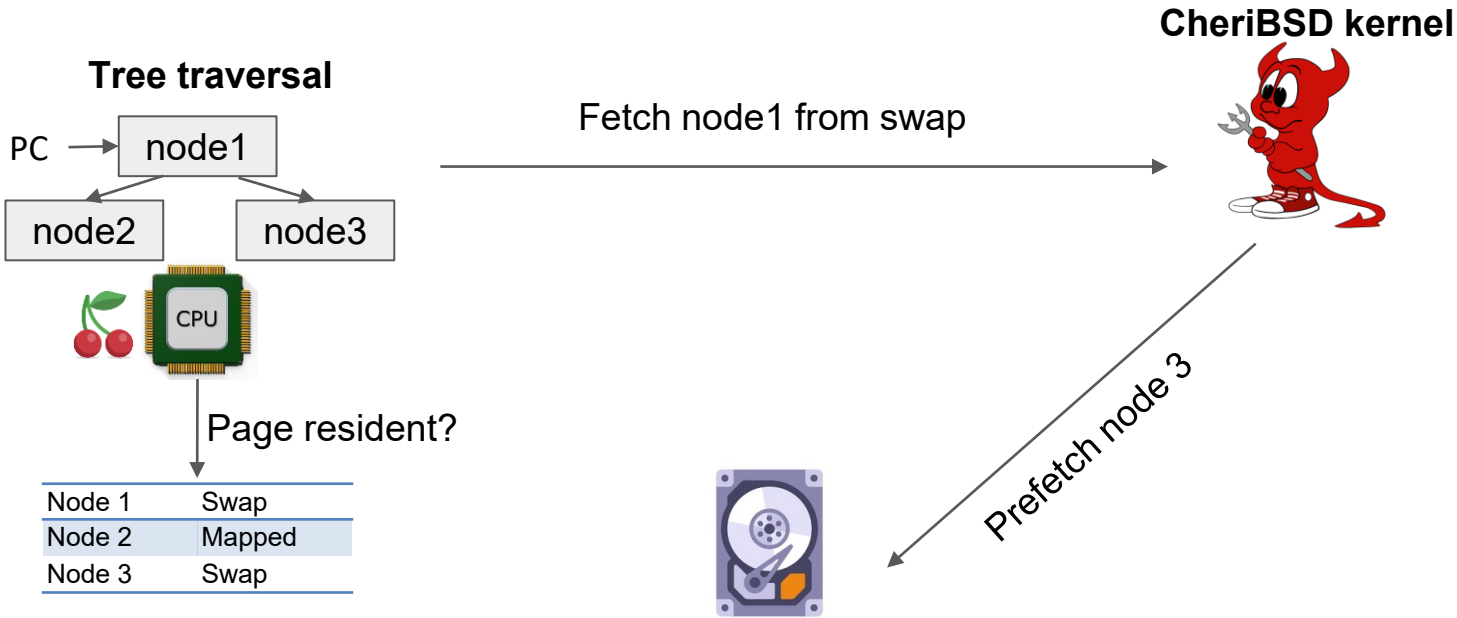
CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.



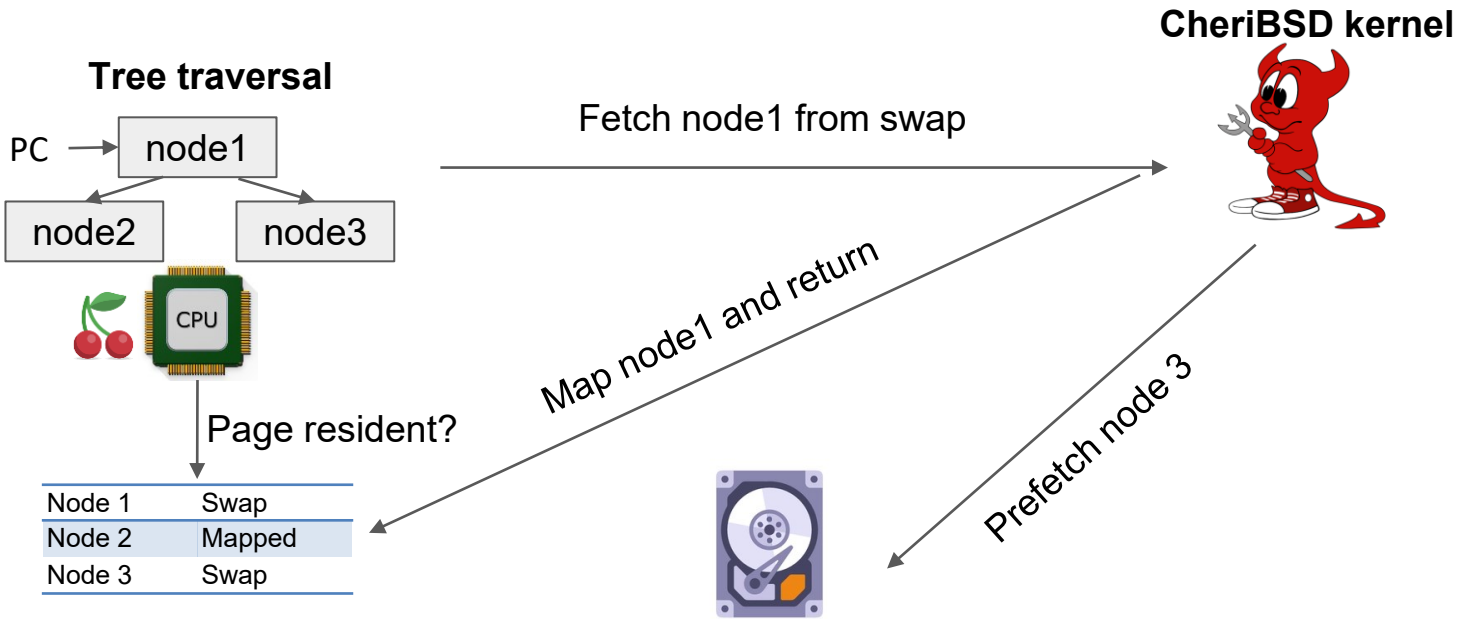
CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.



CHERI-picking overview and design

- CHERI-picking leverages CHERI to make prefetching decisions.
- CHERI treats all pointers as capabilities, that allows the OS to identify pointers in an application agnostic manner.



Evaluation

We implemented CHERI-picking in the **CheriBSD kernel version 22.12**

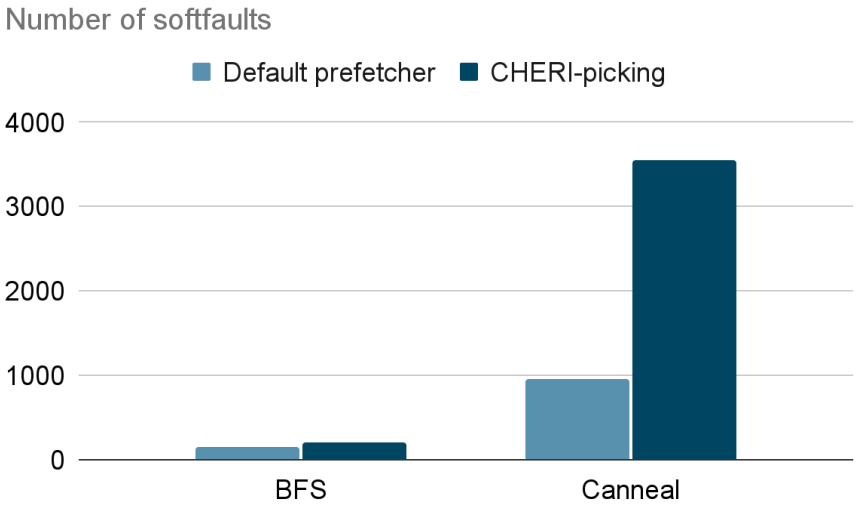
We run evaluations on an **ARM Morello CHERI-capable processor** that contains 4 cores running at 2.4GHz. We limit memory so that the working set size of applications is twice that of the available memory, inducing memory pressure.

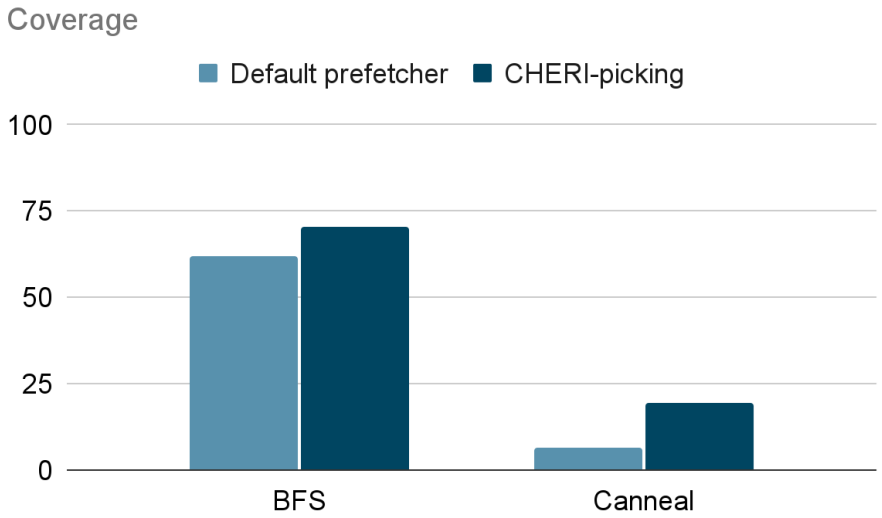
Metrics:

Soft faults: These page faults occur when a page is already in memory, but not mapped into an application's address space; **indicating the prefetcher's prediction capacity.**

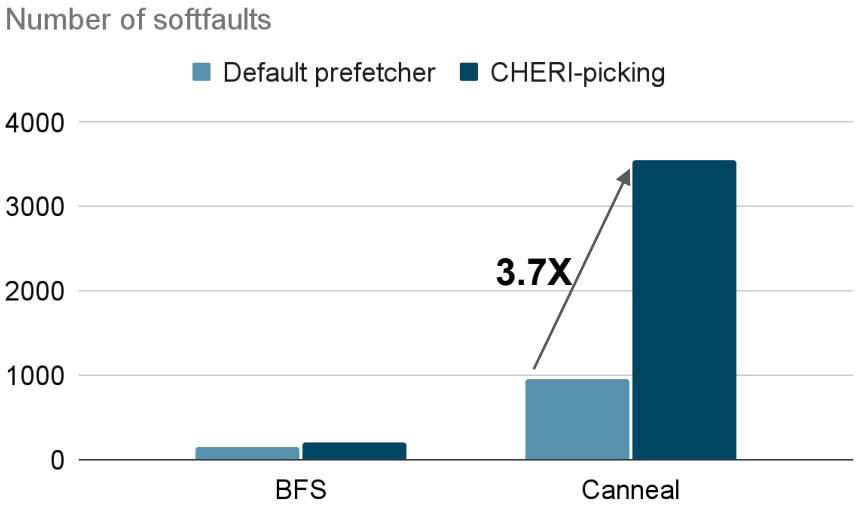
Coverage: The **percentage of page faults that were satisfied** by previously prefetched pages.

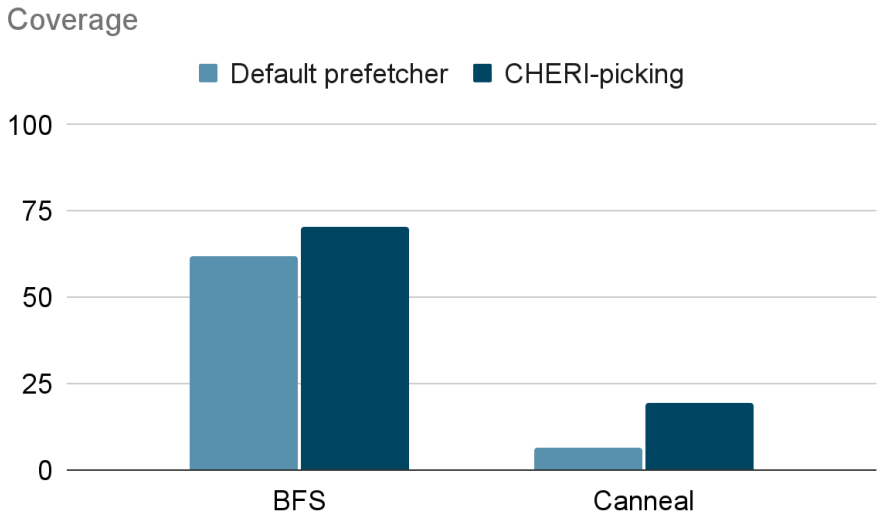
Evaluation


Number of softfaults
Default prefetcher
CHERI-picking
4000
3000
2000
1000
0
BFS
Canneal

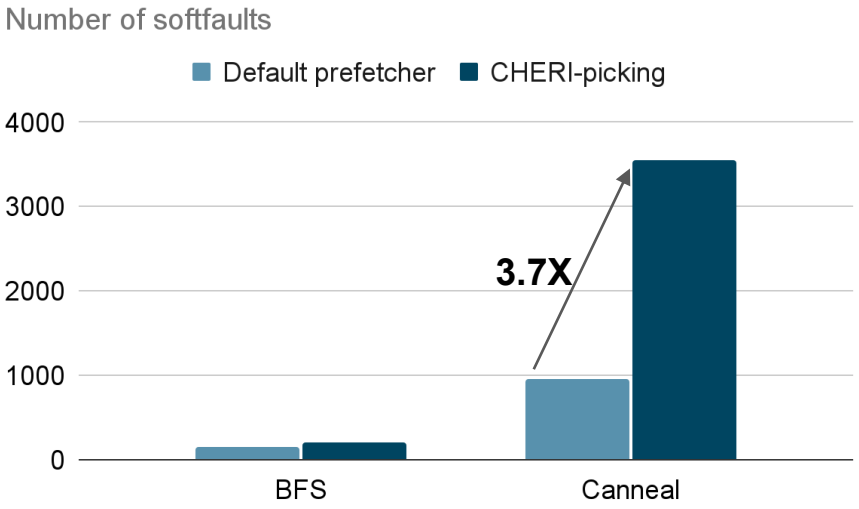

Coverage
Default prefetcher
CHERI-picking
100
75
50
25
0
BFS
Canneal

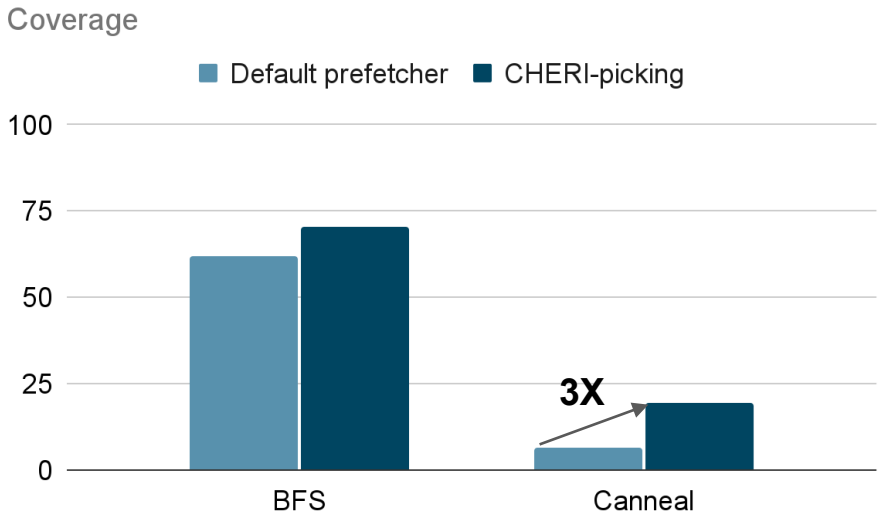
Evaluation


Number of softfaults
Default prefetcher
CHERI-picking
4000
3000
2000
1000
0
3.7X
BFS
Canneal


Coverage
Default prefetcher
CHERI-picking
100
75
50
25
0
BFS
Canneal

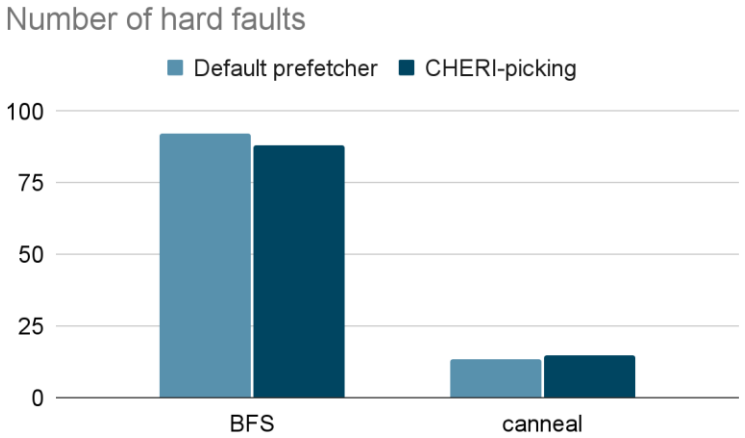
Evaluation


Number of softfaults
Default prefetcher
CHERI-picking
4000
3000
2000
1000
0
3.7X
BFS
Canneal


Coverage
Default prefetcher
CHERI-picking
100
75
50
25
0
3X
BFS
Canneal

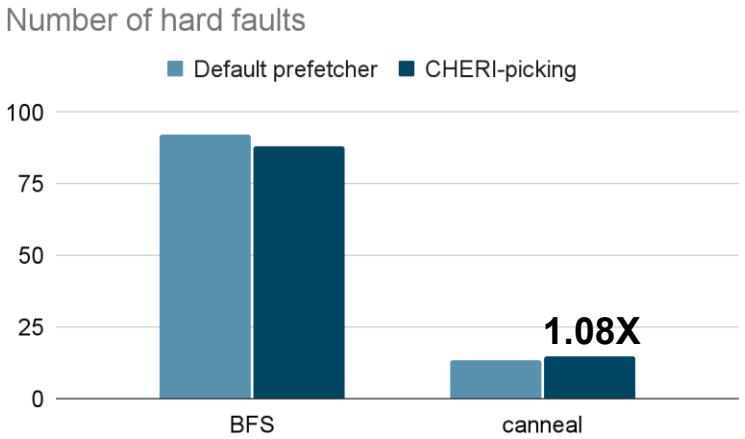
Challenges

- **The CHERI-picking algorithm is naive** and has low accuracy. Major faults are mandatory pagefaults for pages not present in memory.
- **The overhead of running the CHERI-picking algorithm is high** which limits end to end performance improvement.



Challenges

- **The CHERI-picking algorithm is naive** and has low accuracy. Major faults are mandatory pagefaults for pages not present in memory.
- **The overhead of running the CHERI-picking algorithm is high** which limits end to end performance improvement.



Summary

We develop an analyzer and show that applications experience **non-trivial amount of pointer-based pagefaults**.

We introduce CHERI-picking an **application agnostic kernel pointer prefetcher**.

We find that CHERI-picking **improves prefetching coverage by 3X.**

We plan to optimize the CHERI-picking algorithm and improve end to end performance in the future.