LDB: An Efficient Latency Profiling Tool for Multithreaded Applications

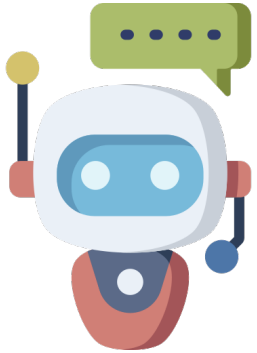
Inho Cho* Seo Jin Park † Ahmed Saeed ‡
Mohammad Alizadeh* Adam Belay*

* MIT CSAIL † USC University of Southern California ‡ Georgia Tech

Latency-sensitive Applications

Web

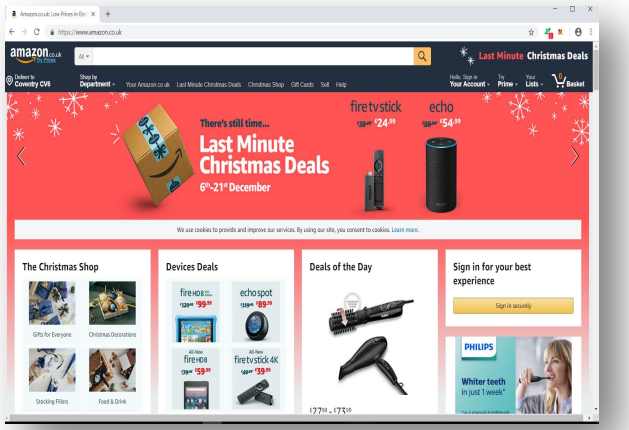
Real-time Ad Bidding



Interactive ML inference

High-frequency stock trading

High Fan-out



Web

Recommender

AD

AD

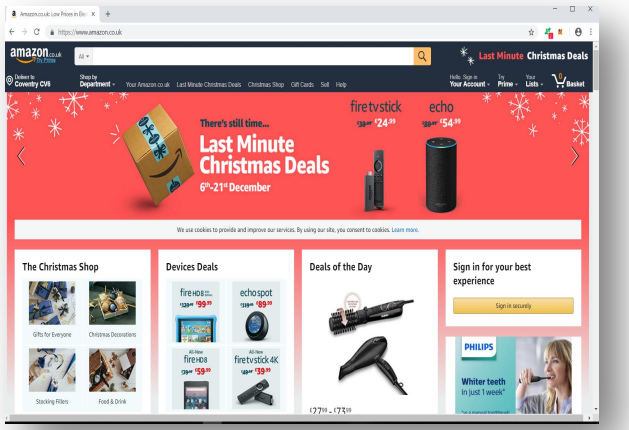
AD

Ad

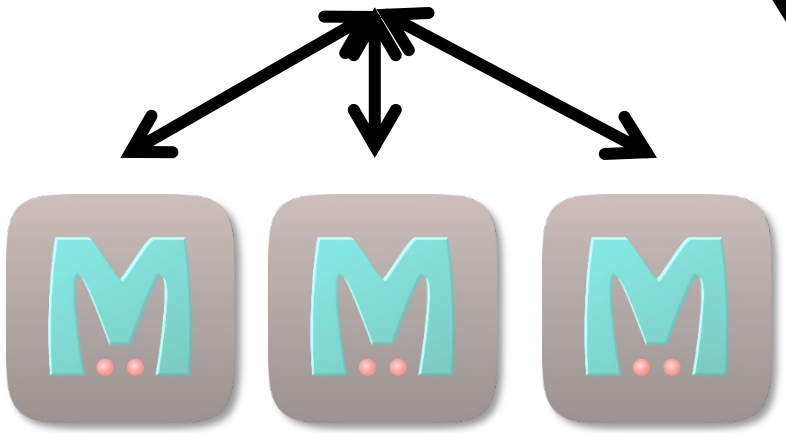
Database

Cache

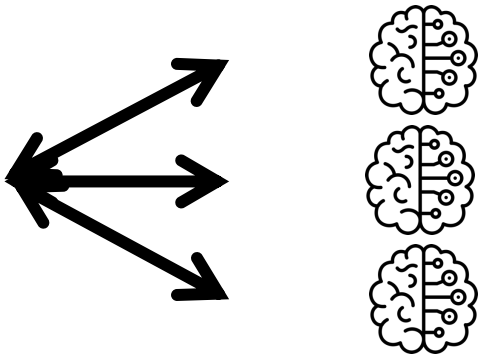
Low RPC Tail Latency is Critical



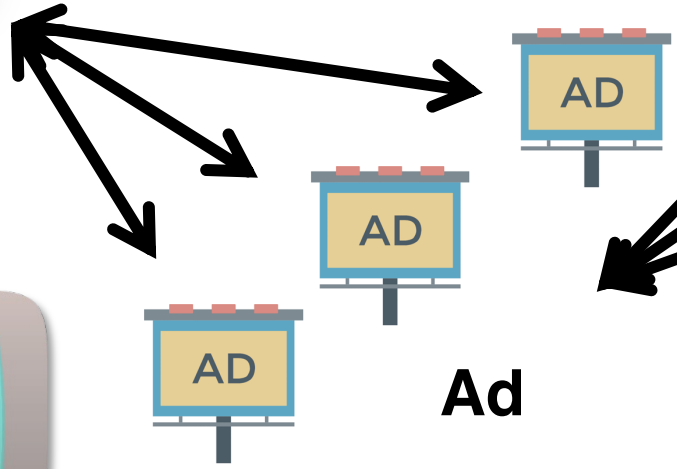
Web



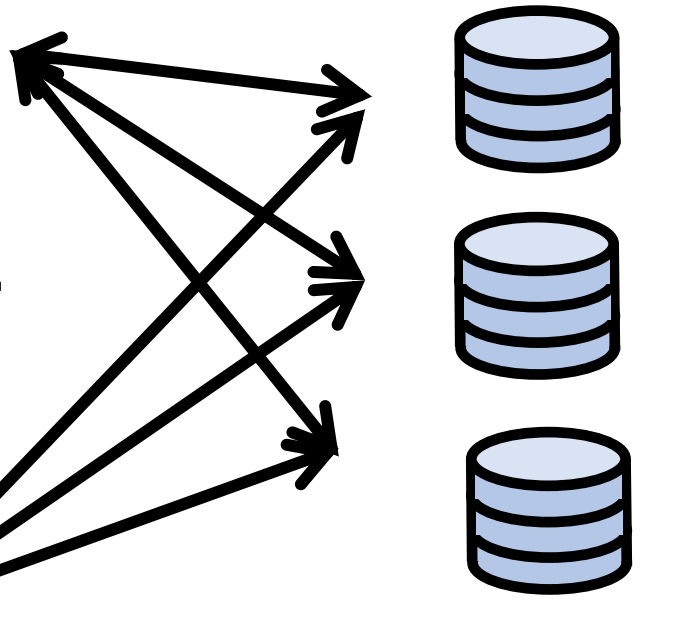
Cache



Recommender

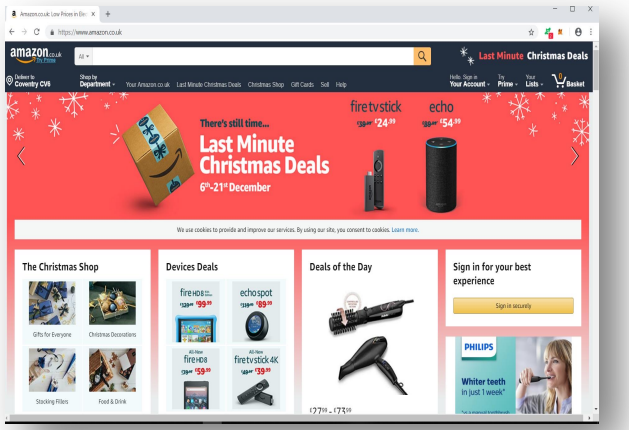


Ad



Database

Tail Latency in Networks



Web

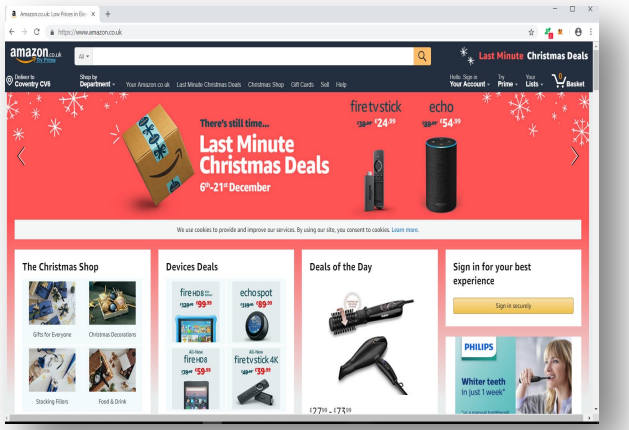
Recommender

AD

AD

Apache Skywalking, Jaeger, Naver Pinpoint, HindSight[NSDI'23], DeepFlow[SIGCOMM'23], BufScope [NSDI'22], SpiderMon [SOSR'20], etc.

Tail Latency at Applications



Web

Recommender

AD

AD

AD

Ad

Database

Cache

Simple Key-Value Store Application

```
std::map<int, std::string> db;
pthread_mutex_t lock;
```

Simple Key-Value Store Application

```
std::map<int, std::string> db;
pthread_mutex_t lock;

int request_handler (
          int key, std::string& value) {
    pthread_mutex_lock(&lock);
    db[key] = value;
    pthread_mutex_unlock(&lock);
}
```

Simple Key-Value Store Application

```
std::map<int, std::string> db;
pthread_mutex_t lock;

int request_handler (
          int key, std::string& value) {
    pthread_mutex_lock(&lock);
    db[key] = value;
    pthread_mutex_unlock(&lock);
}
```

```
int background_thread() {
    while (true) {
        snapshot();
        usleep(10000);
    }
}
```

Simple Key-Value Store Application

```
std::map<int, std::string> db;
pthread_mutex_t lock;

int request_handler (
          int key, std::string& value) {
    pthread_mutex_lock(&lock);
    db[key] = value;
    pthread_mutex_unlock(&lock);
}
```

```
int background_thread() {
    while (true) {
        snapshot();
        usleep(10000);
    }
}

int snapshot() {
    std::ofstream
        out("snapshot.txt");
    pthread_mutex_lock(&lock);
    for (const auto& kv : db)
        out << kv.first << ","
            << kv.second
            << std::endl;
    pthread_mutex_unlock(&lock);
    out.close();
}
```

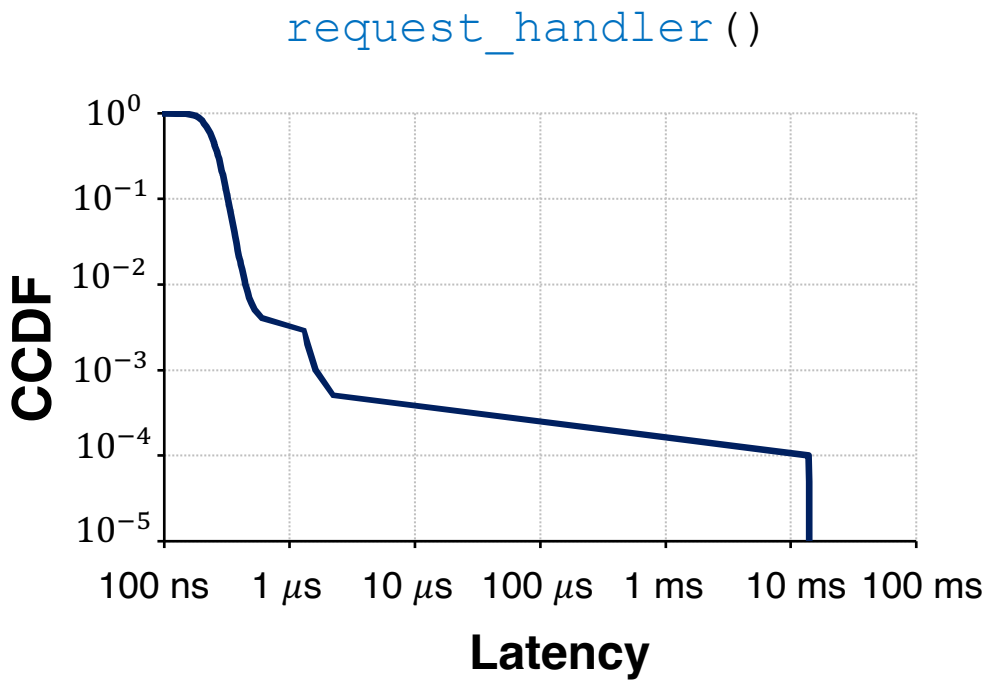
Simple Key-Value Store Application

```
std::map<int, std::string> db;
pthread_mutex_t lock;

int request_handler (
        int key, std::string& value) {
    pthread_mutex_lock(&lock);
    db[key] = value;
    pthread_mutex_unlock(&lock);
}
```

```
int background_thread() {
    while (true) {
        snapshot();
        usleep(10000);
    }
}

int snapshot() {
    std::ofstream
        out("snapshot.txt");
    pthread_mutex_lock(&lock);
    for (const auto& kv : db)
        out << kv.first << ","
            << kv.second
            << std::endl;
    pthread_mutex_unlock(&lock);
    out.close();
}
```

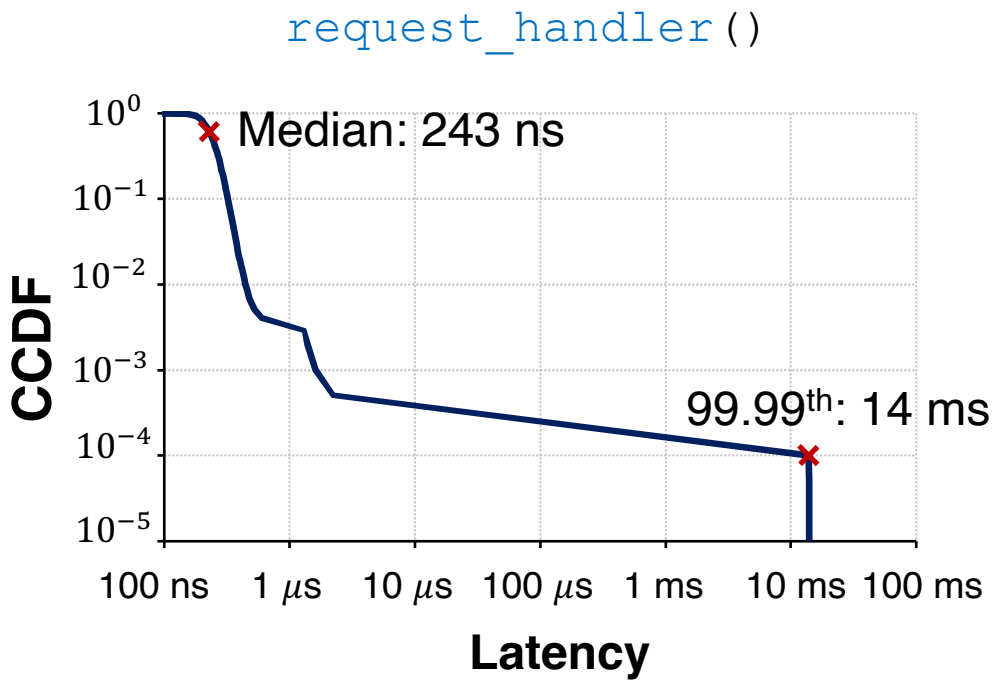
High Dispersion in Request Processing Time

```
std::map<int, std::string> db;
pthread_mutex_t lock;

int request_handler (
        int key, std::string& value) {
    pthread_mutex_lock(&lock);
    db[key] = value;
    pthread_mutex_unlock(&lock);
}
```



High Dispersion in Request Processing Time

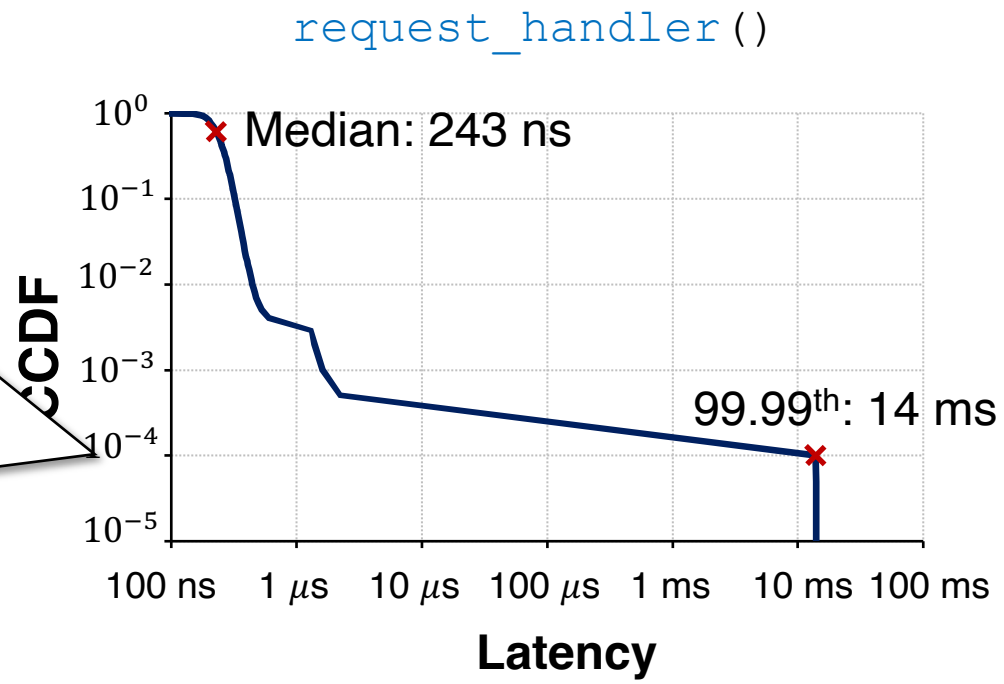
```
std::map<int, std::string> db;
pthread_mutex_t lock;

int request_handler (
        int key, std::string& value) {
    pthread_mutex_lock(&lock);
    db[key] = value;
    pthread_mutex_unlock(&lock);
}
```



High Dispersion in Request Processing Time

```
std::map<int, std::string> db;
pthread_mutex_t lock;
```

What is happening every 10,000 requests?



Strawman: Using CPU Profiler (perf, gperftools)

Strawman: Using CPU Profiler (perf, gperftools)

Function	CPU Time
`generate_random_string`	63.75 %
`request_handler`	7.43 %
`std::_Rb_tree_increment`	2.82 %
… (13 more functions) …	
`snapshot`	0.62%

Strawman: Using CPU Profiler (perf, gperftools)

Function	CPU Time
`generate_random_string`	63.75 %
`request_handler`	7.43 %
`std::_Rb_tree_increment`	2.82 %
… (13 more functions) …	
`snapshot`	0.62%

Strawman: Using CPU Profiler (perf, gperftools)

Function	CPU Time
`generate_random_string`	63.75 %
`request_handler`	7.43 %
`std::_Rb_tree_increment`	2.82 %
… (13 more functions) …	
`snapshot`	0.62%

Doesn't detect snapshot as the cause of high tail latency.

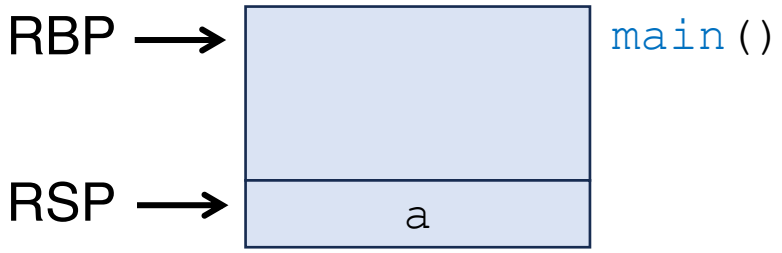
LDB: Efficient Tail Latency Profiling Tool

Live Demo

x86 Stack Frame


```
int main() {


}
```

x86 Stack Frame

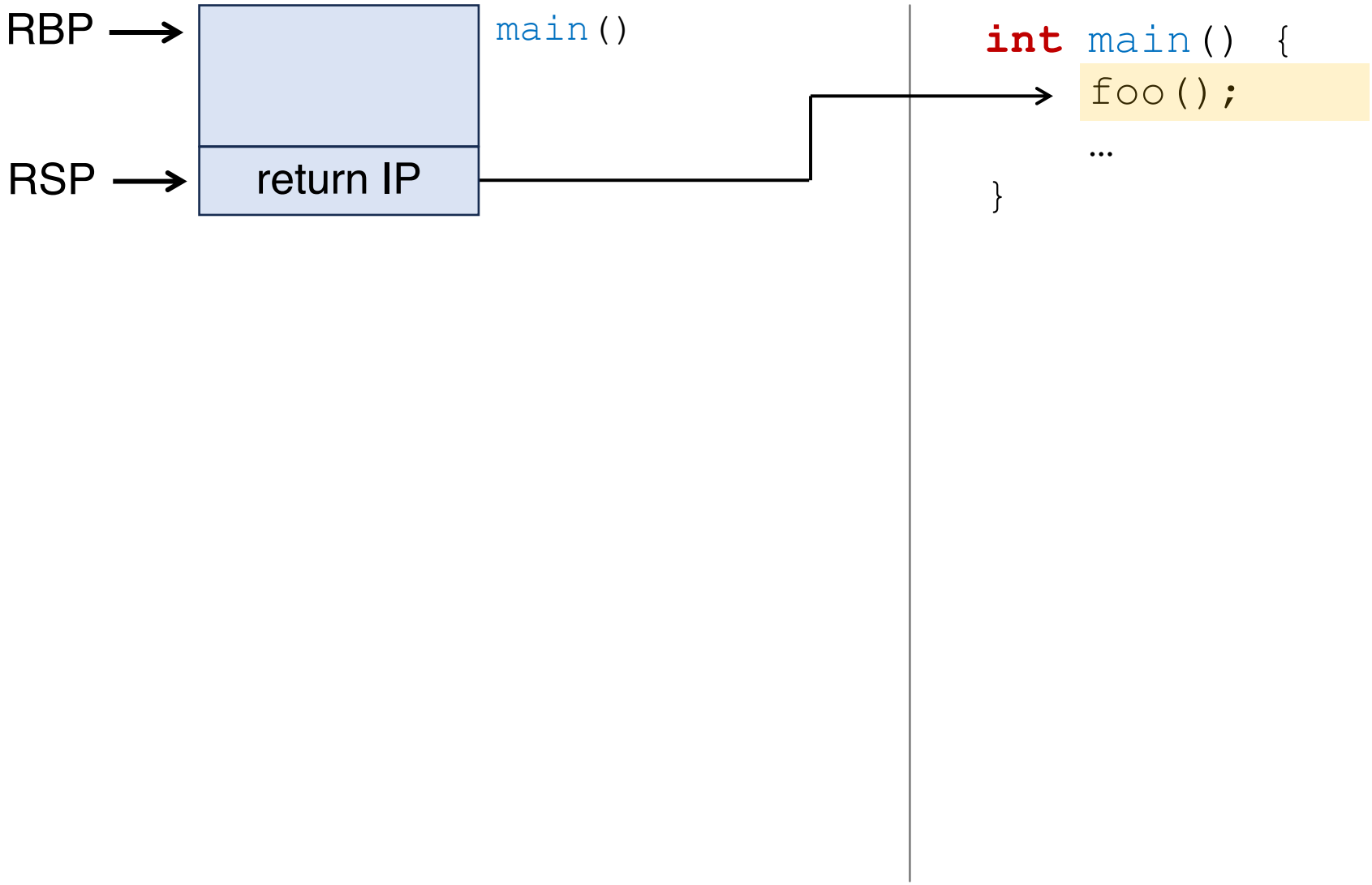



```
int main() {
    int a;

}
```

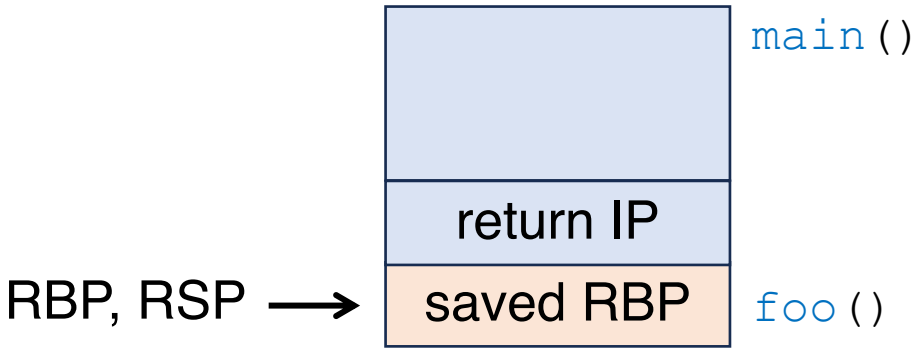
x86 Stack Frame


```
int main() {
    foo();
    …
}
```

x86 Stack Frame

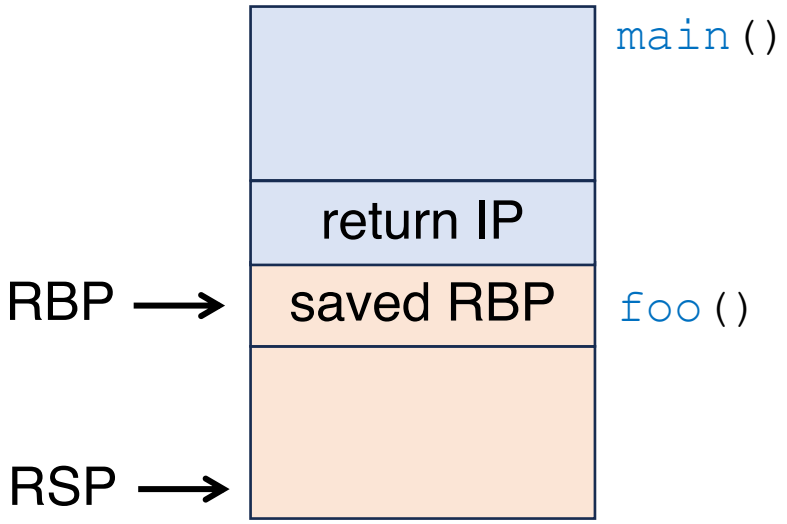


x86 Stack Frame



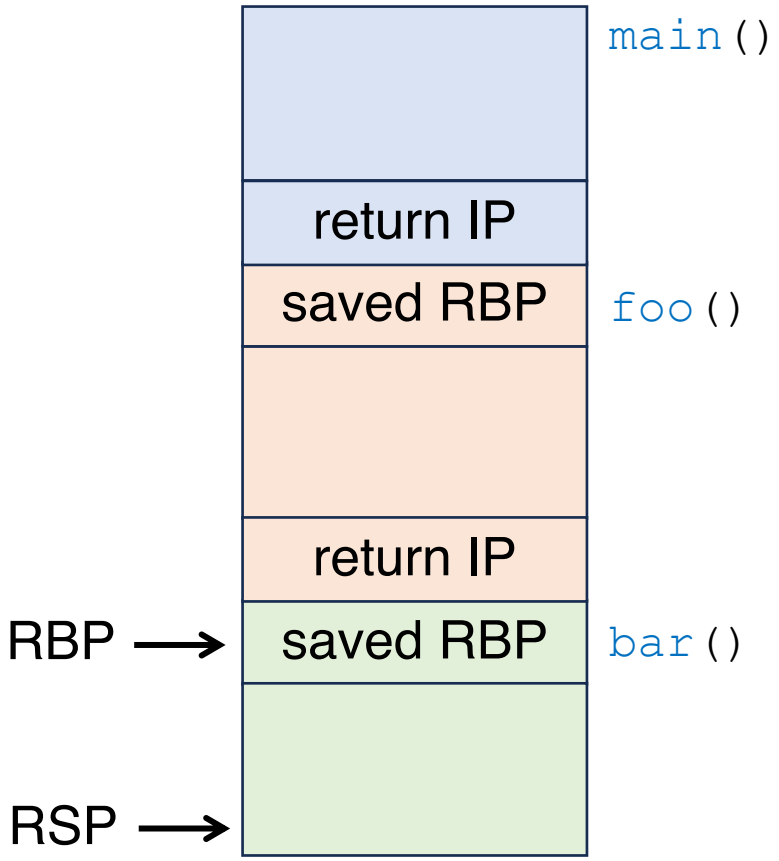

```
int main() {
    foo();
    …
}

int foo() {


}
```

x86 Stack Frame



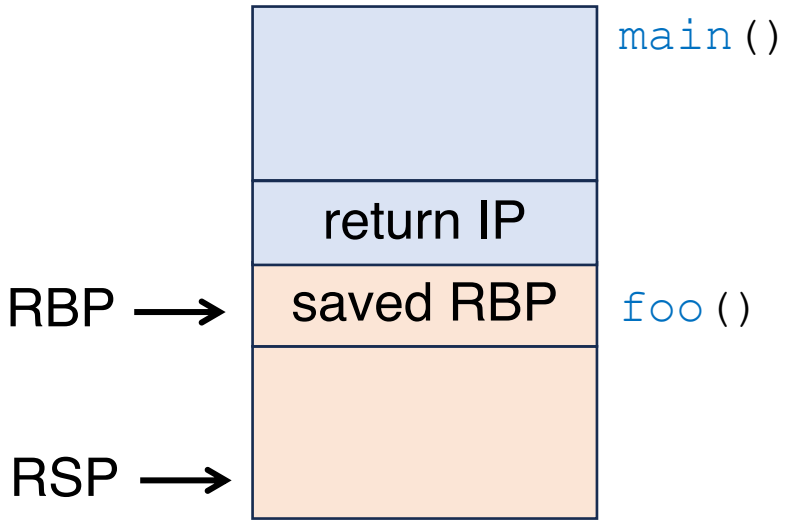

```
int main() {
    foo();
    …
}

int foo() {
    for (i = 0; i < 2; ++i)
        bar();
}
```

x86 Stack Frame



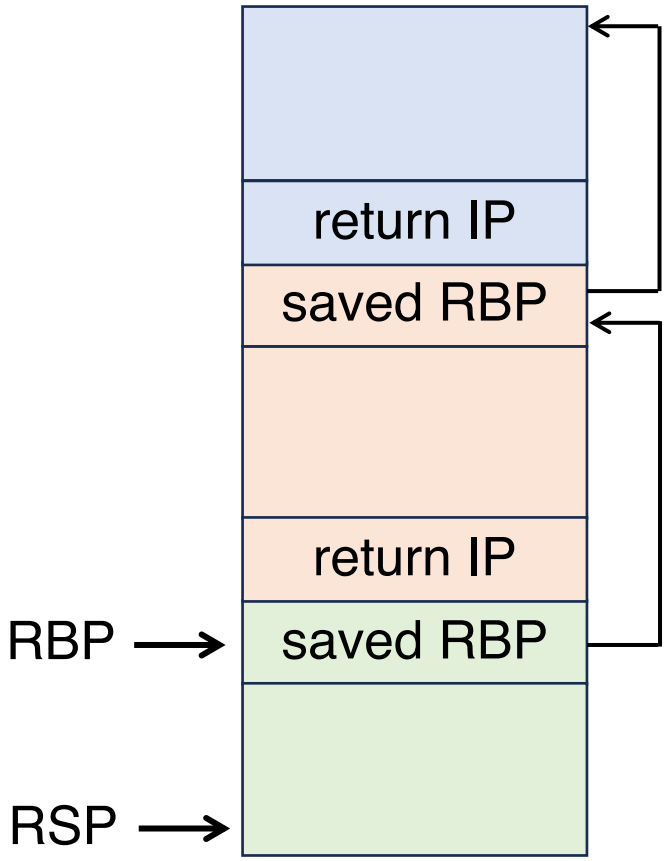

```
int main() {
    foo();
    …
}

int foo() {
    for (i = 0; i < 2; ++i)
        bar();
}

int bar() {
    …
}
```

x86 Stack Frame



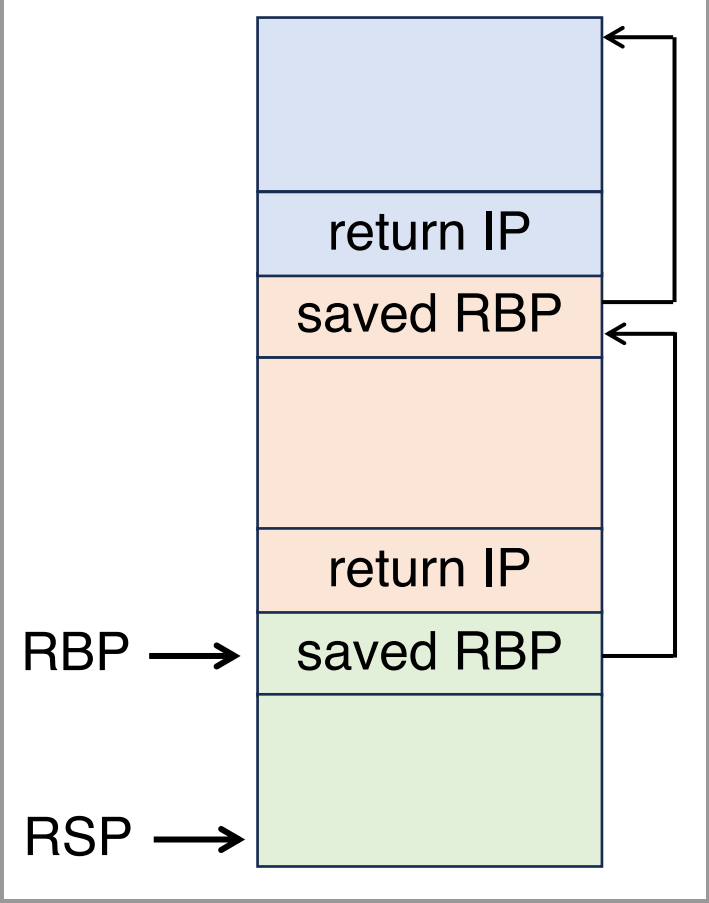

```
int main() {
    foo();
    …
}

int foo() {
    for (i = 0; i < 2; ++i)
        bar();
}
```

x86 Stack Frame




```
int main() {
    foo();
    …
}

int foo() {
    for (i = 0; i < 2; ++i)
        bar();
}

int bar() {
    …
}
```

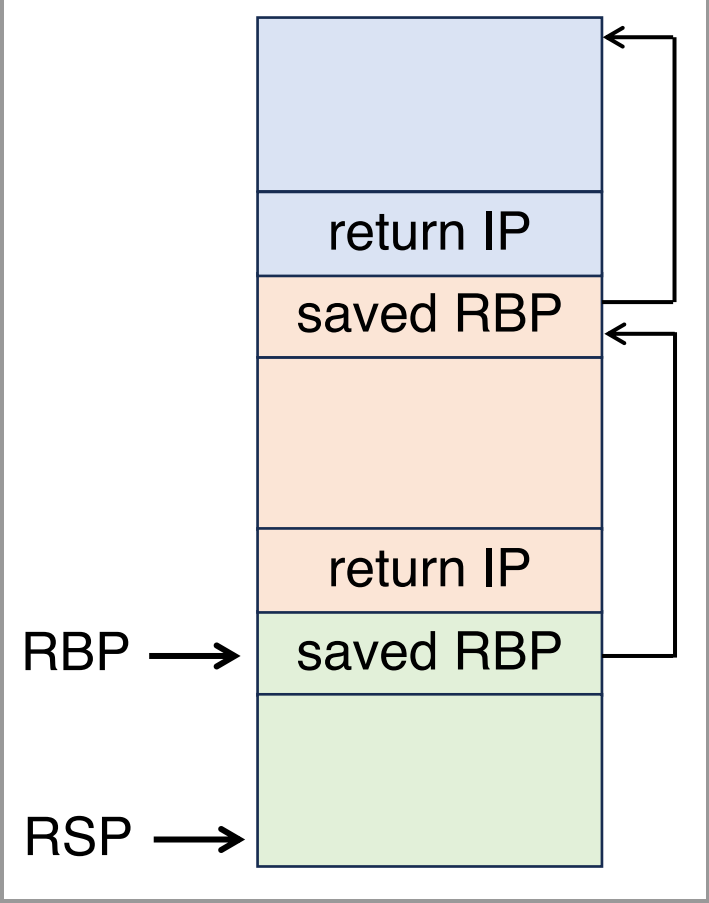
x86 Stack Frame



Intuition

The metadata in stack frames remains the same as long as the thread is stuck with a bottleneck function

x86 Stack Frame



Intuition

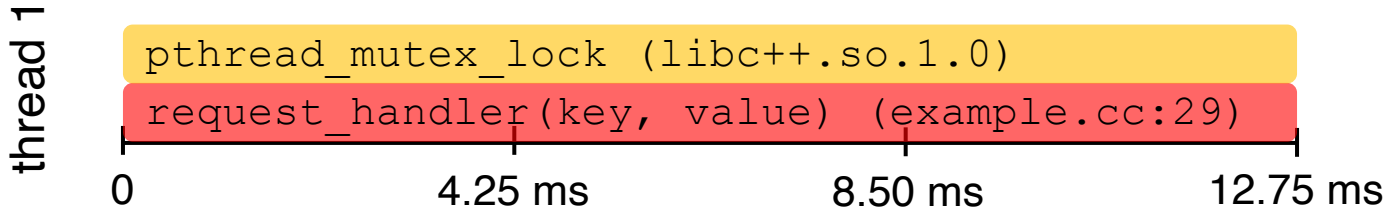
The metadata in stack frames remains the same as long as the thread is stuck with a bottleneck function

Let's sample them!

Accurate and Useful Profiling Results

LDB can profile application's **tail latency behavior** with

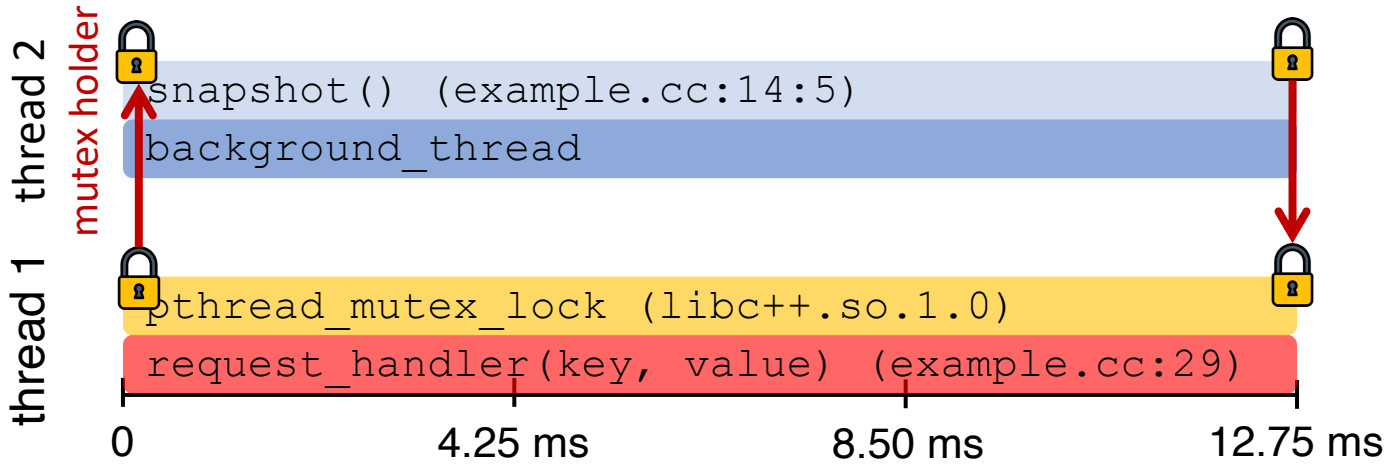
(1) High accuracy including inter-thread communication



Accurate and Useful Profiling Results

LDB can profile application's **tail latency behavior** with

(1) High accuracy including inter-thread communication

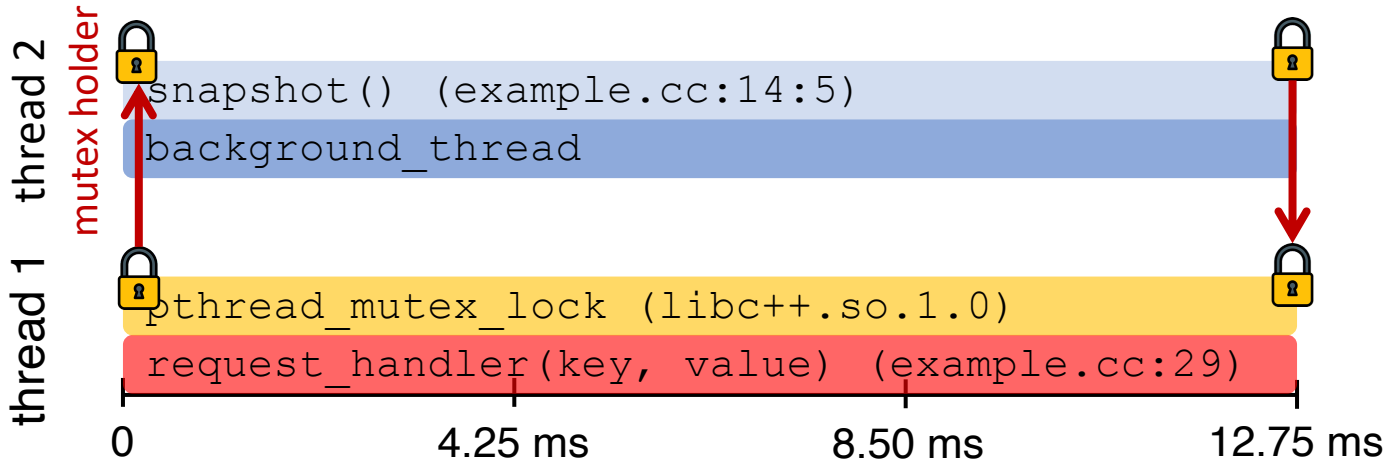


Low Runtime Overhead

LDB can profile application's **tail latency behavior** with

(1) High accuracy including inter-thread communication

(2) Reasonable runtime overhead



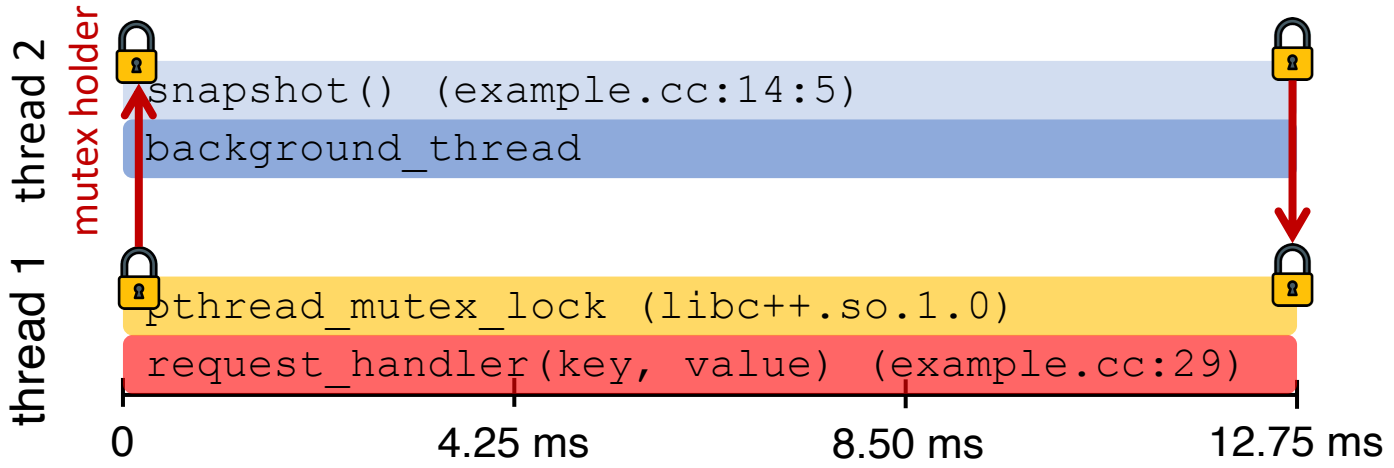
Interactive Debugging with Fast Result

LDB can profile application's **tail latency behavior** with

(1) High accuracy including inter-thread communication

(2) Reasonable runtime overhead

(3) Interactive debugging



Great Portability

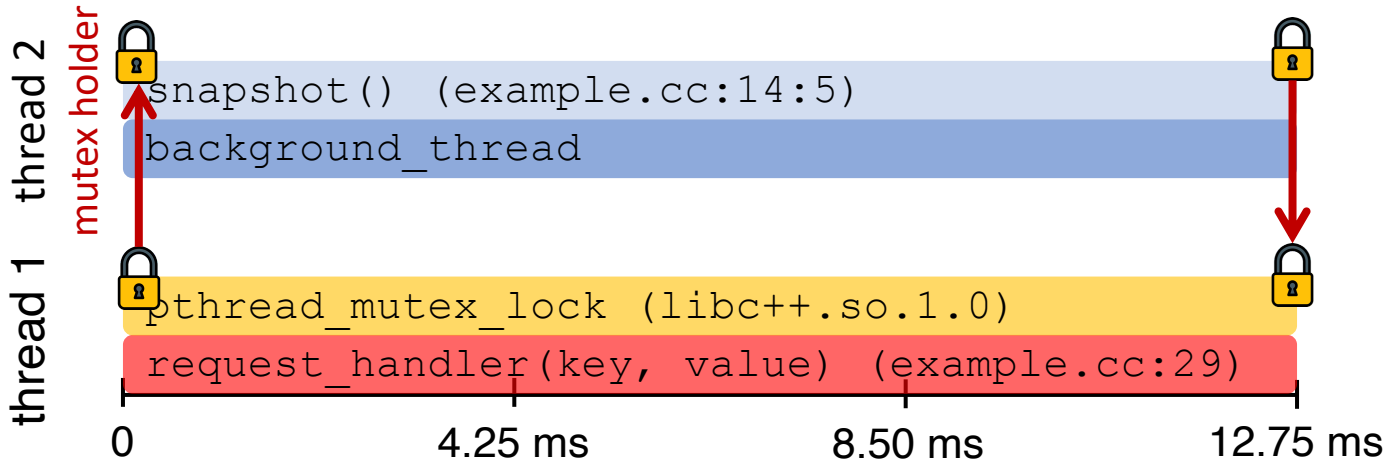
LDB can profile application's **tail latency behavior** with

(1) High accuracy including inter-thread communication

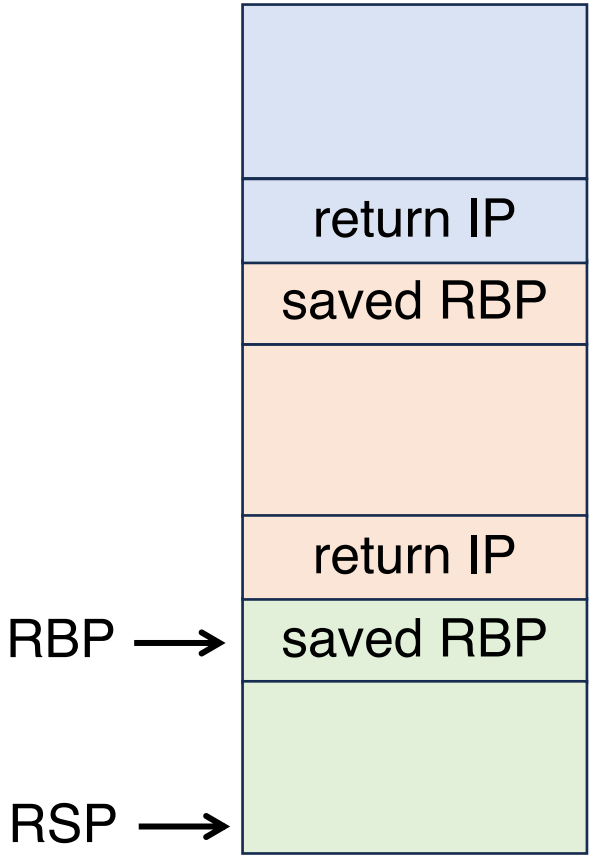
(2) Reasonable runtime overhead

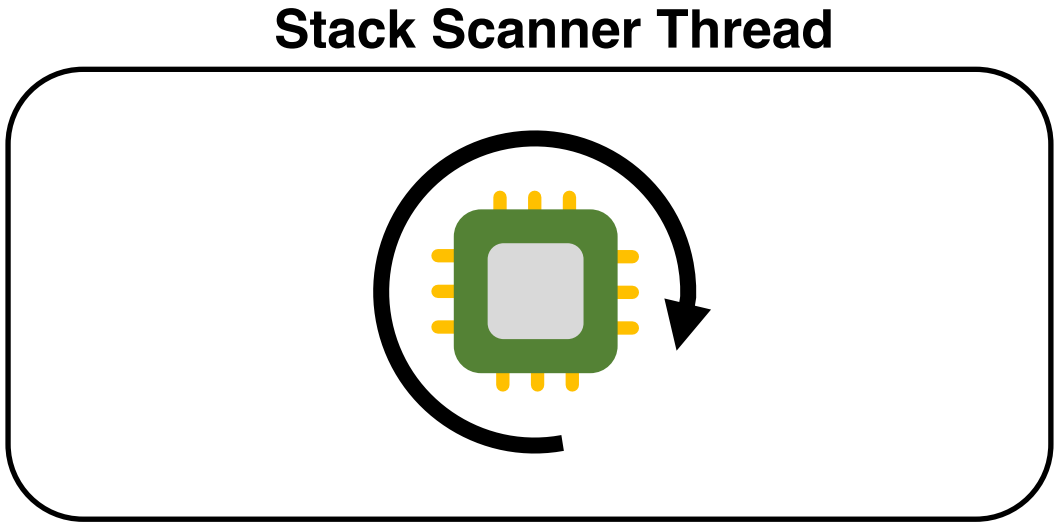
(3) Interactive debugging

(4) Great portability

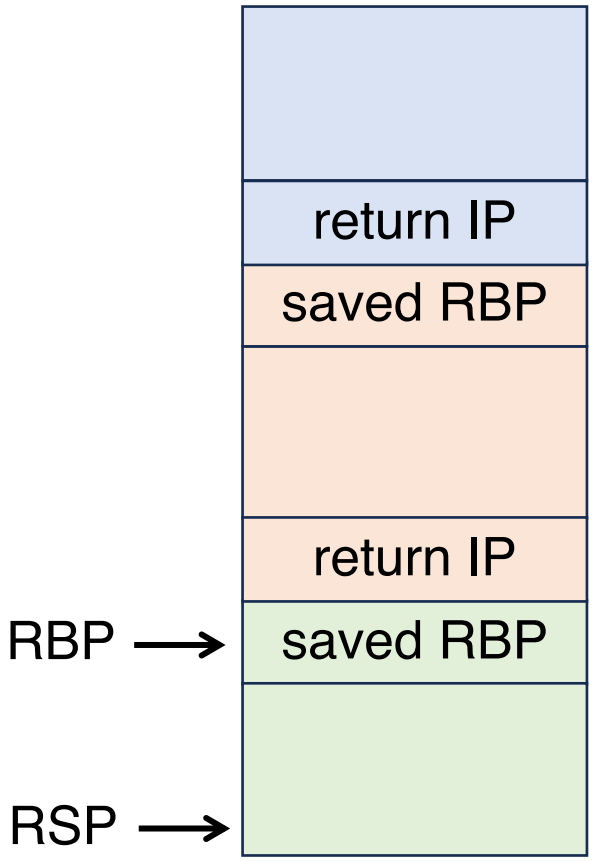


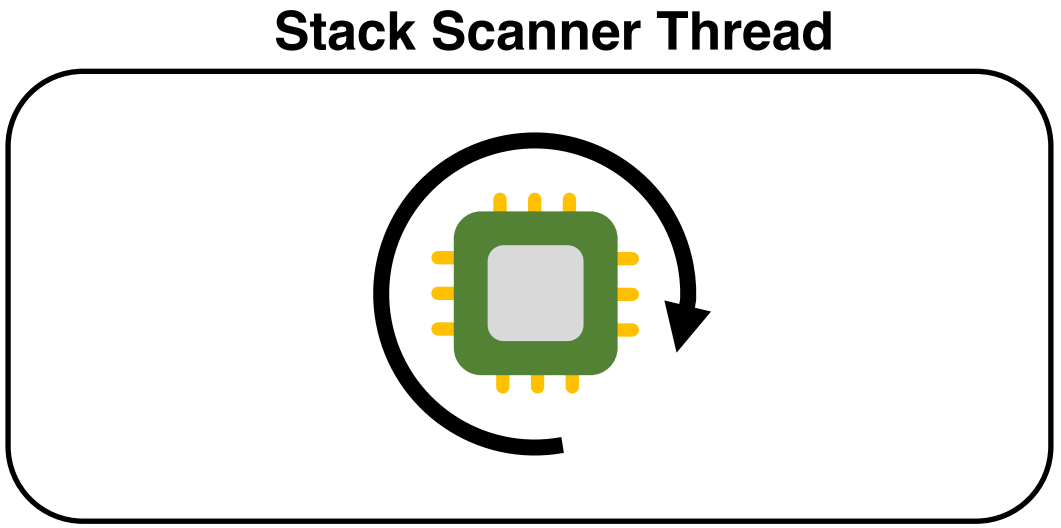
Stack Scanner Thread



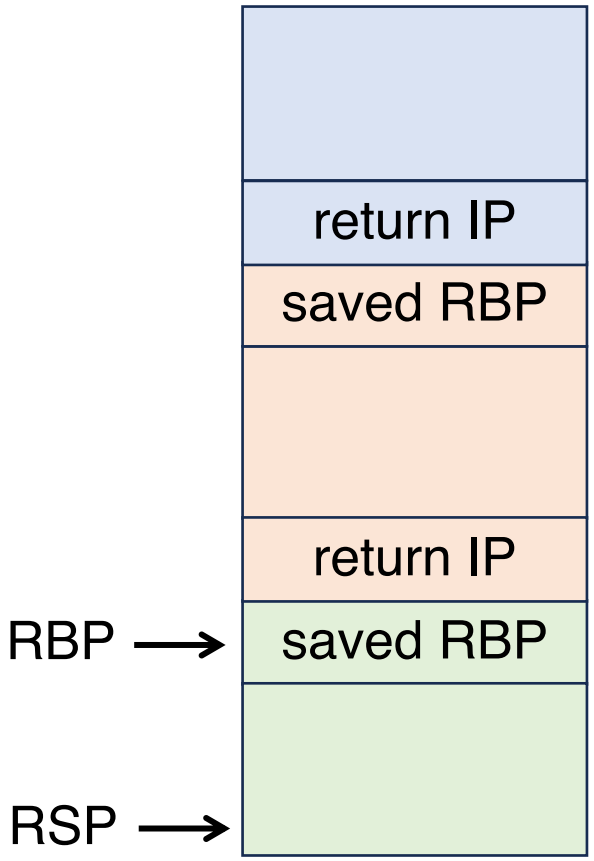


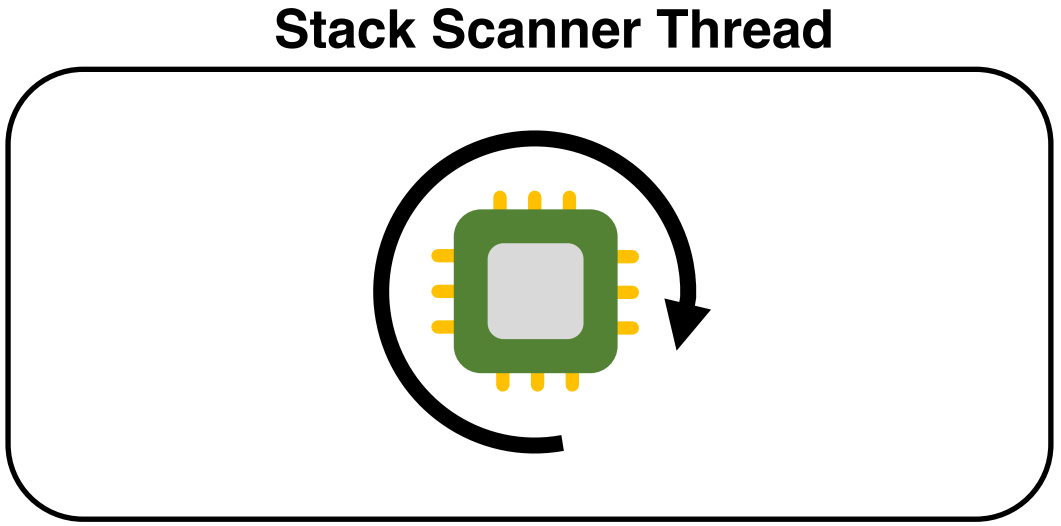
Challenge #1. The Most Recent RBP

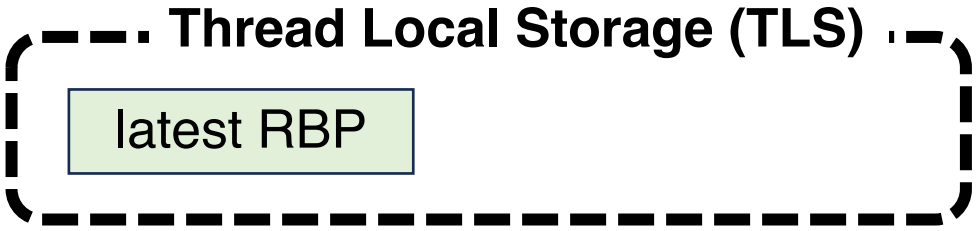




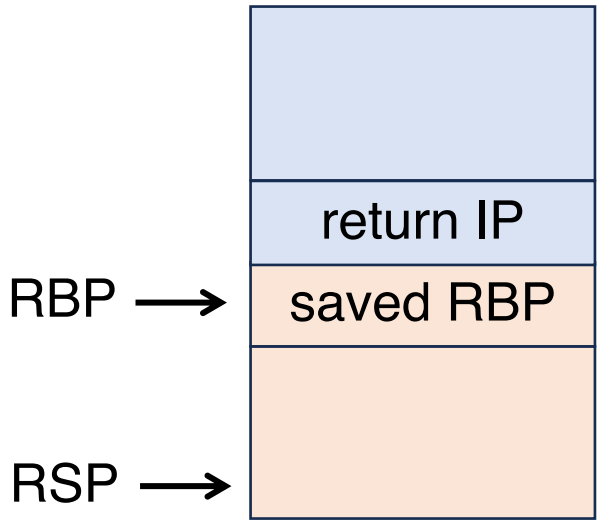
Latest RBP in Thread Local Storage

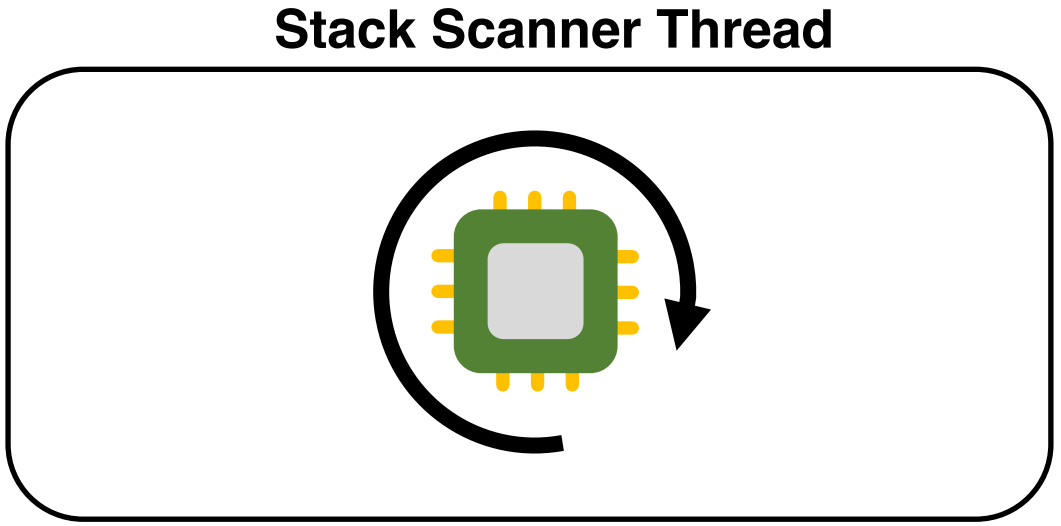


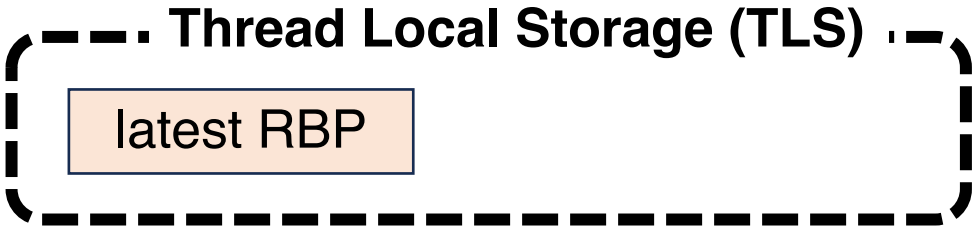




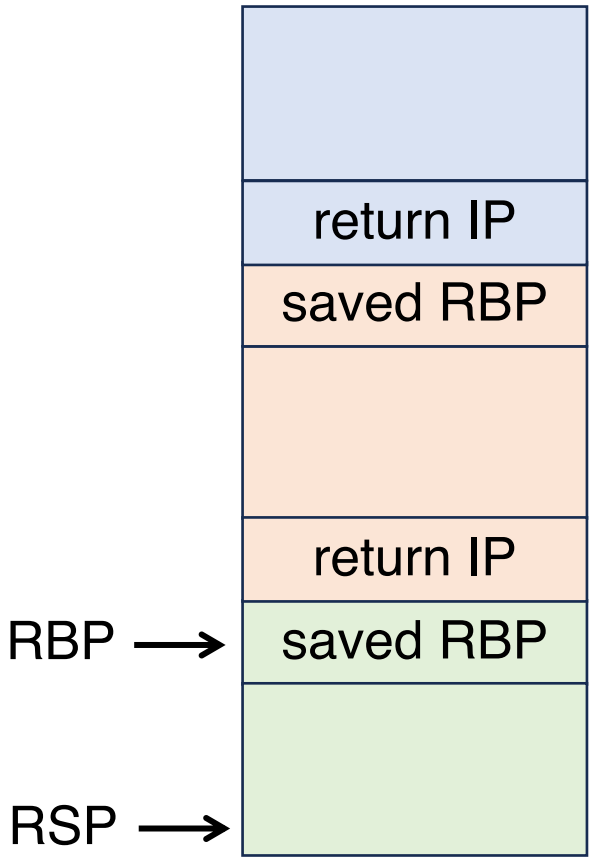
Latest RBP in Thread Local Storage

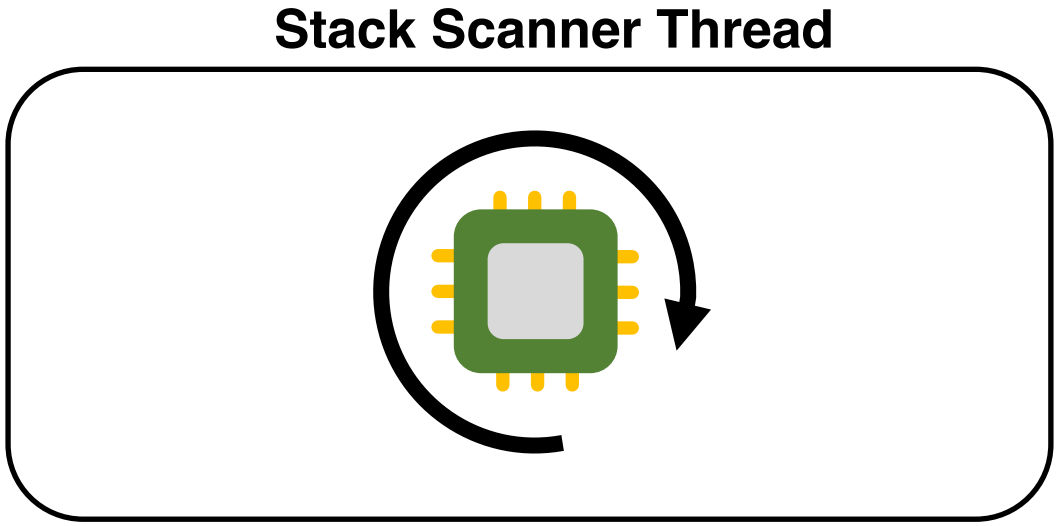


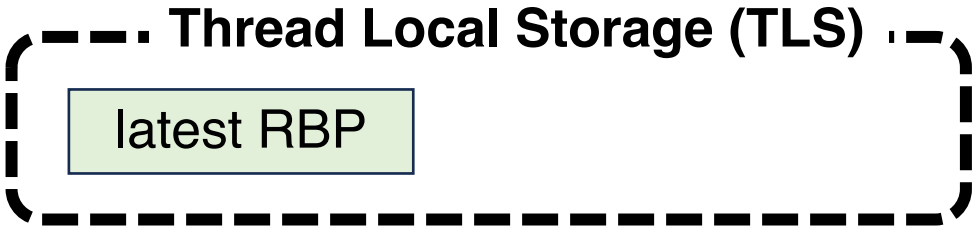




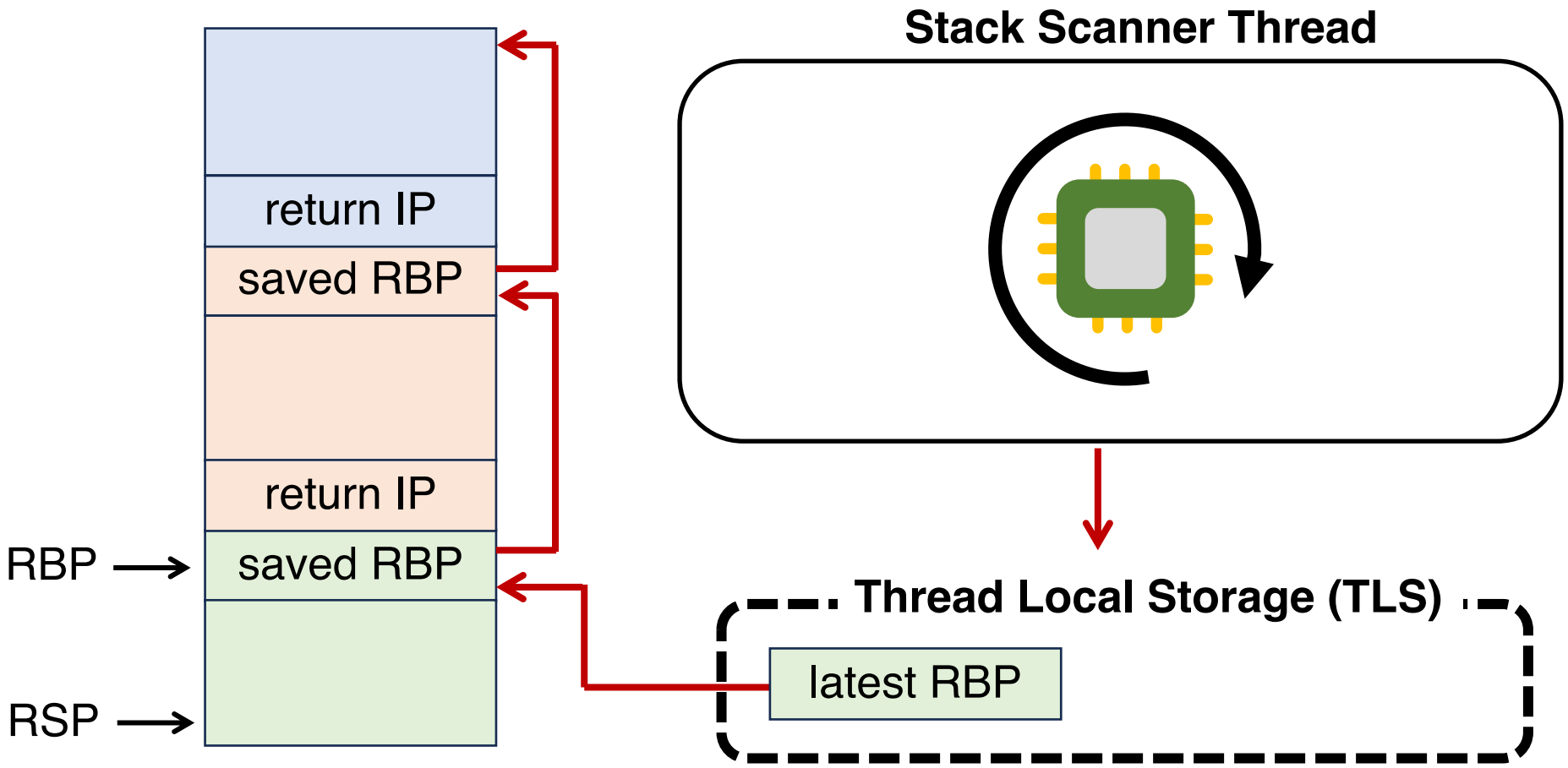
Latest RBP in Thread Local Storage



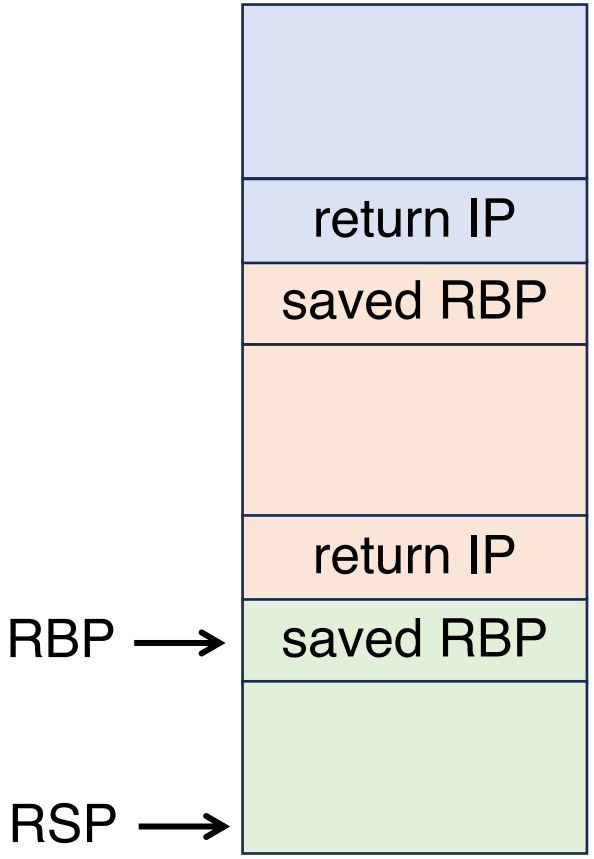




Stack Sampling

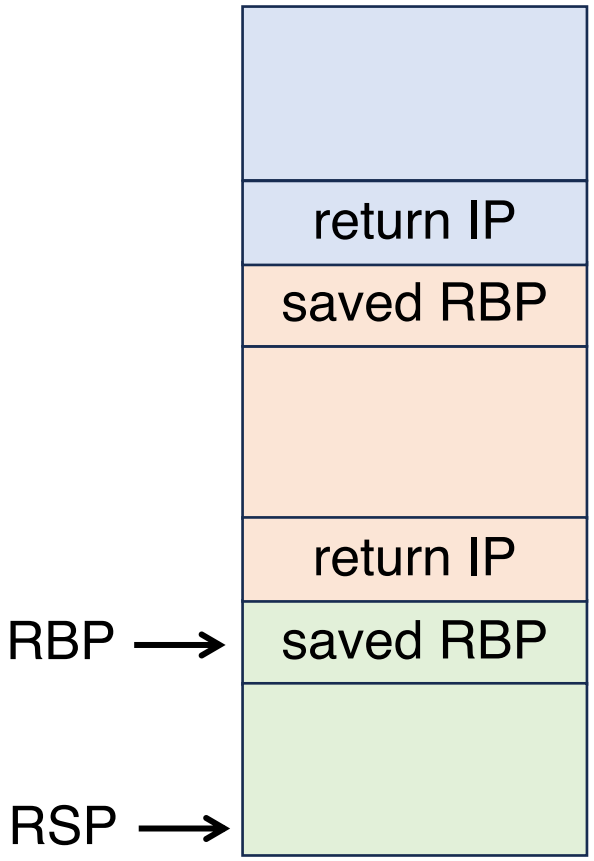


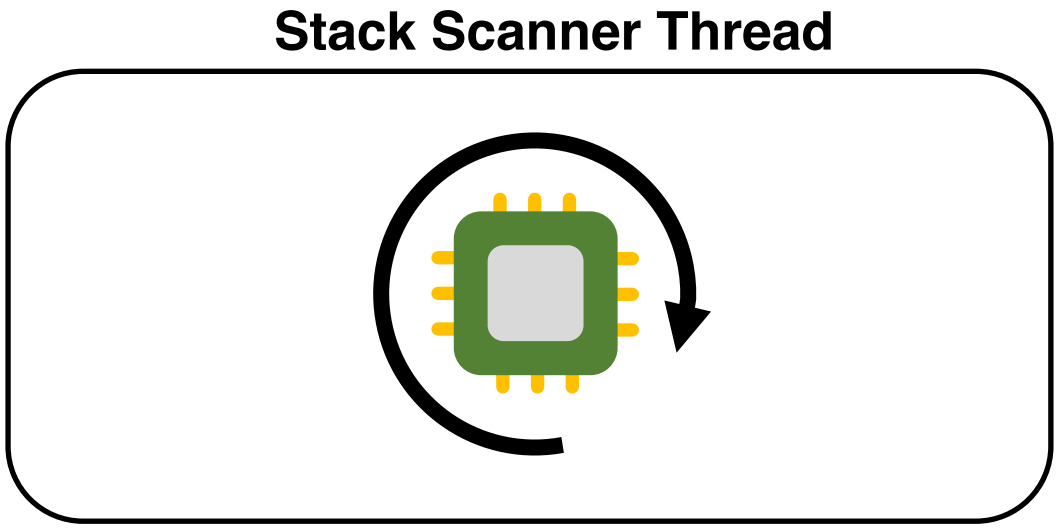
Challenge #2. Differentiating Function Calls

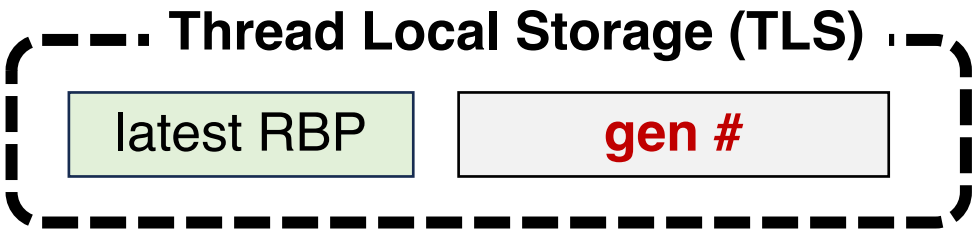



```
int main() {
    foo();
    …
}

int foo() {
    for (i = 0; i < 2; ++i)
        bar();
}

int bar() {
    …
}
```

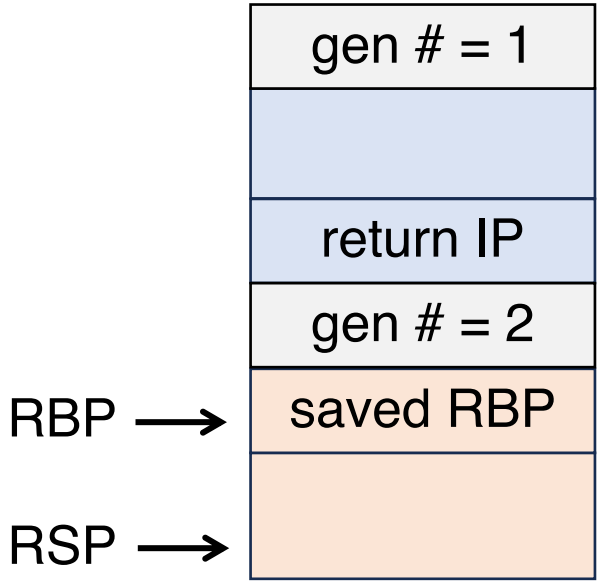
Generation Number

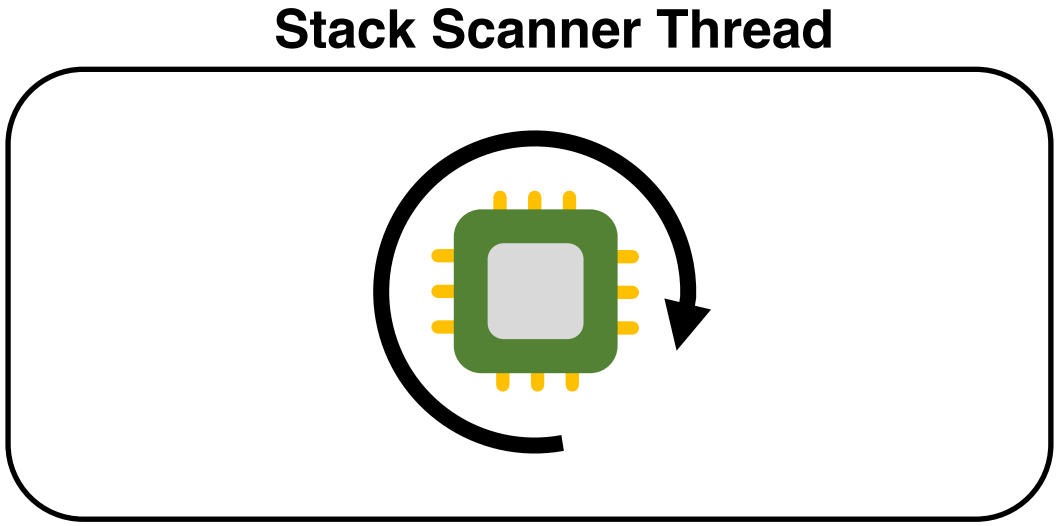




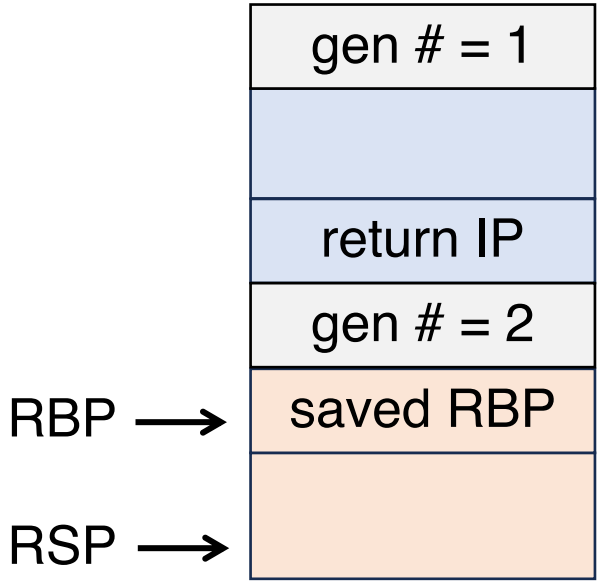


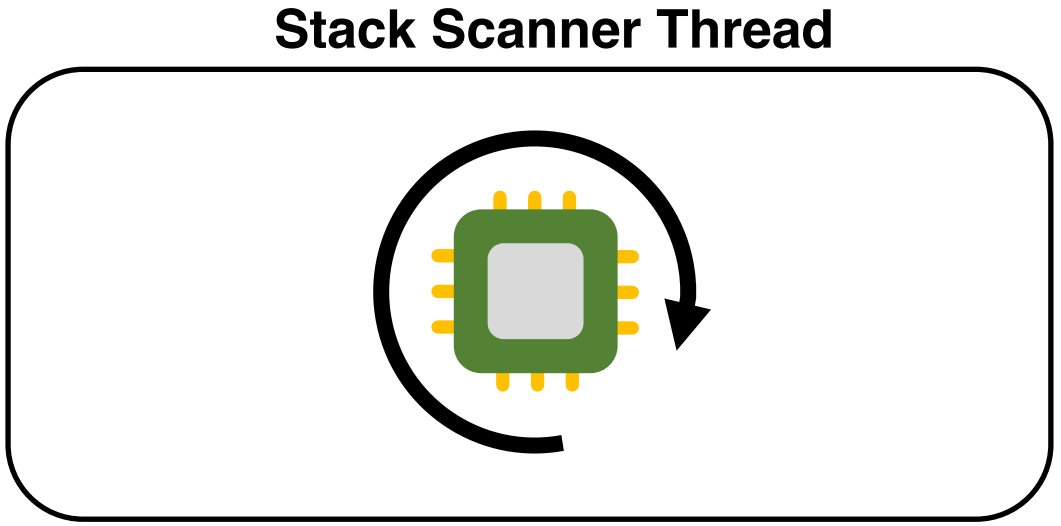
Generation Number



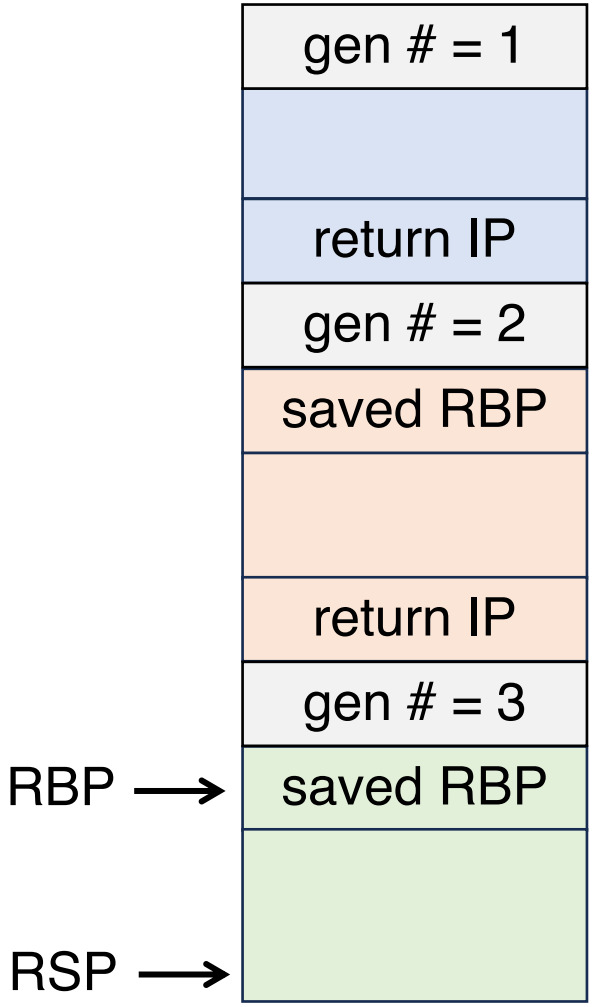


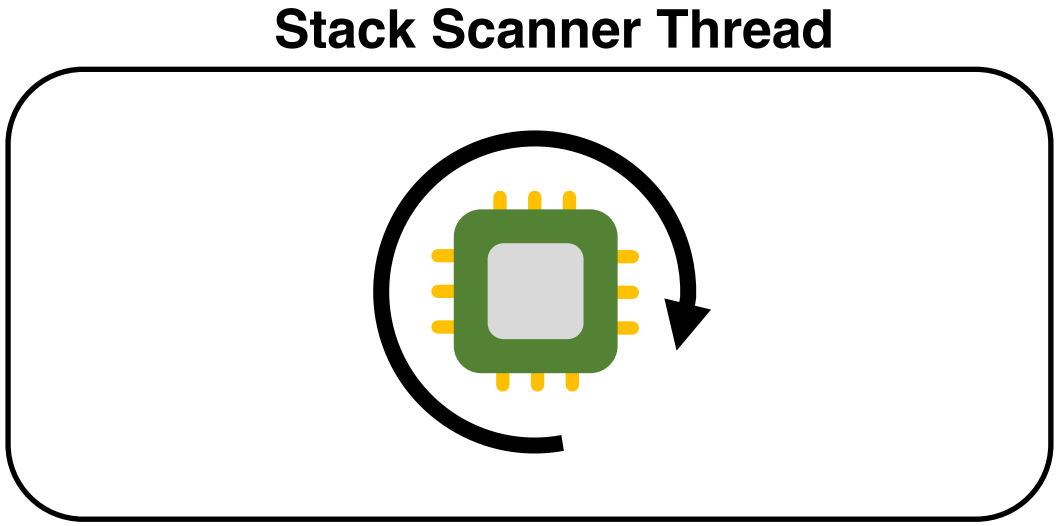
Generation Number



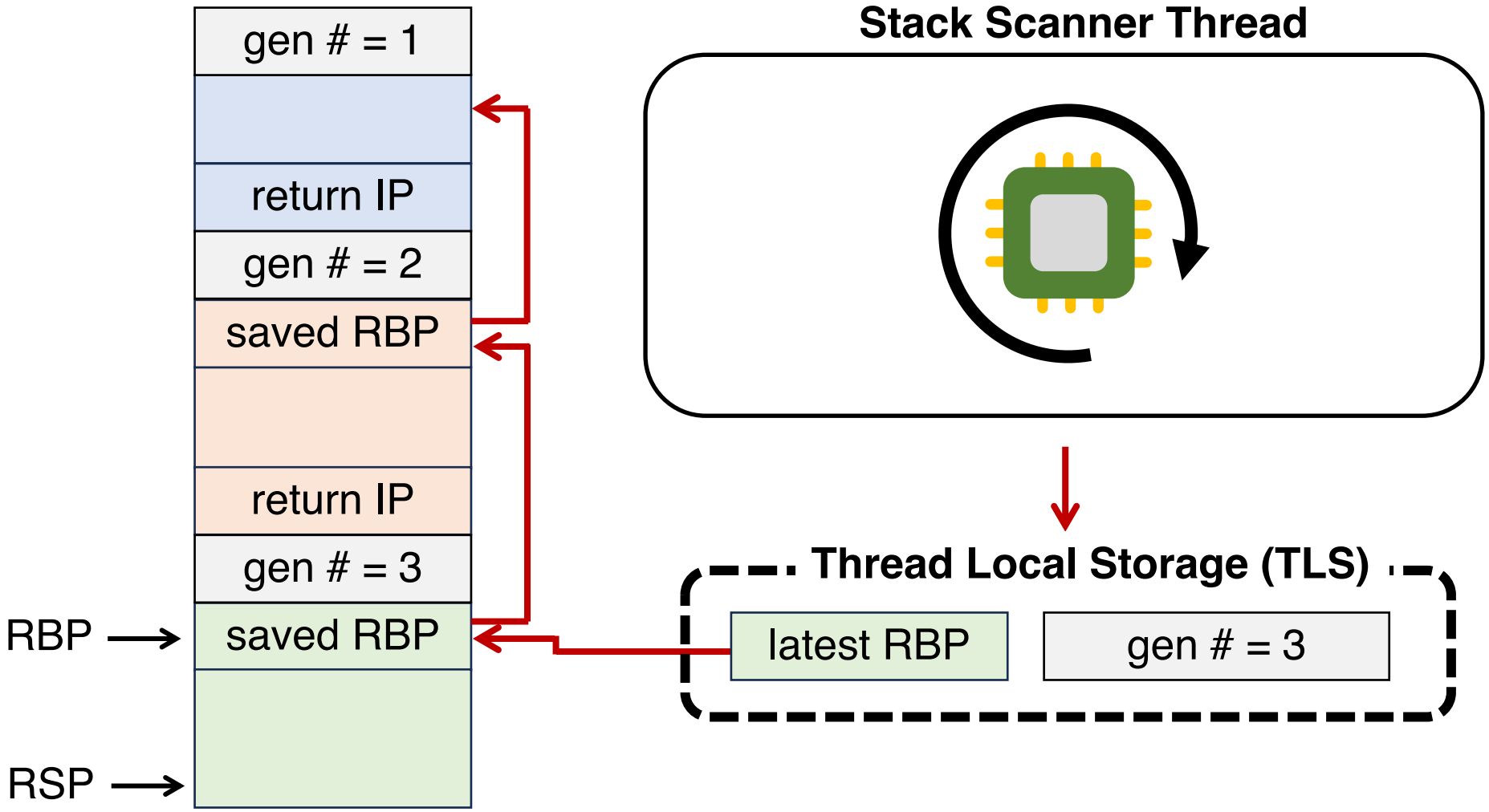


Generation Number

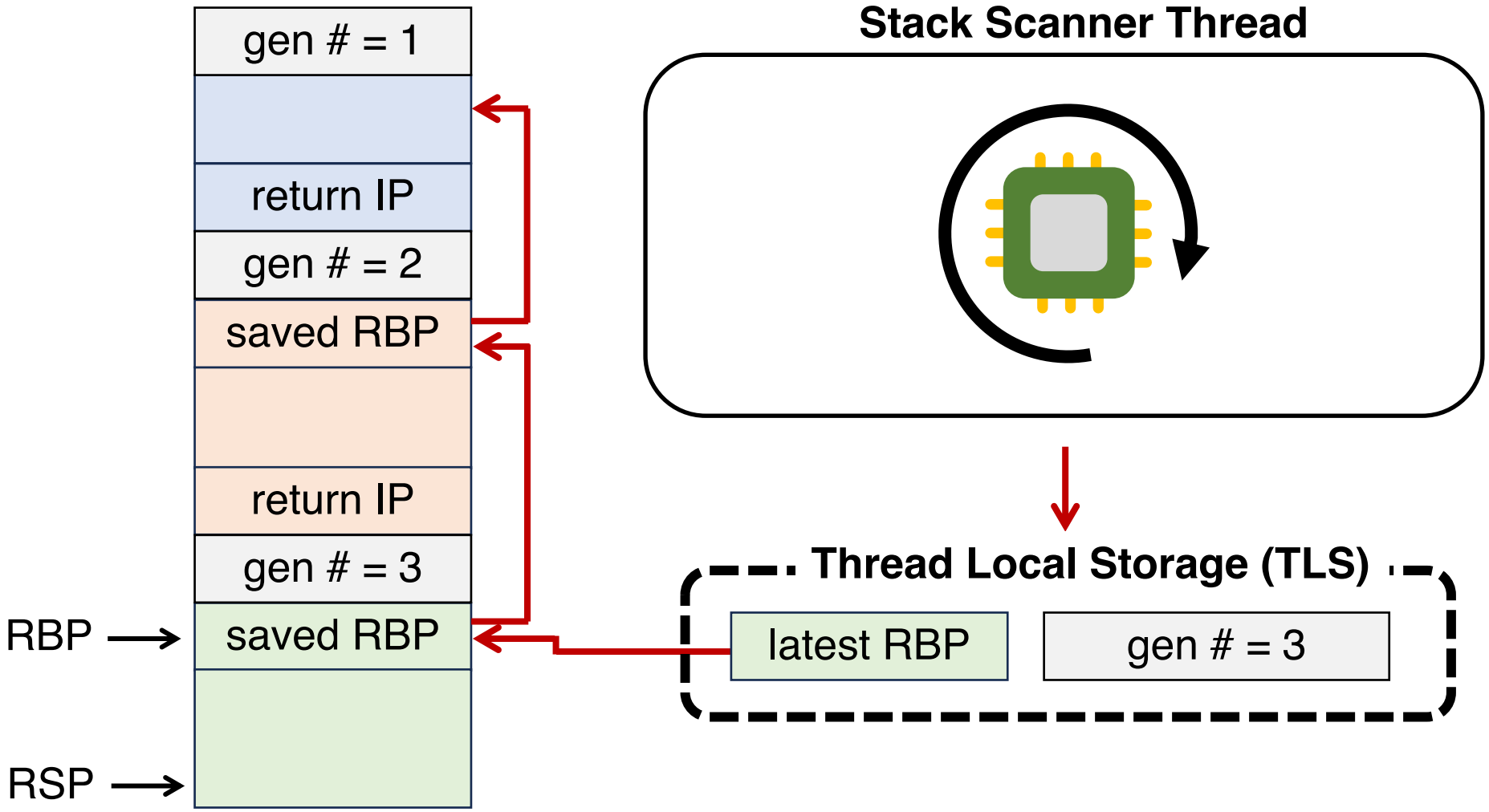




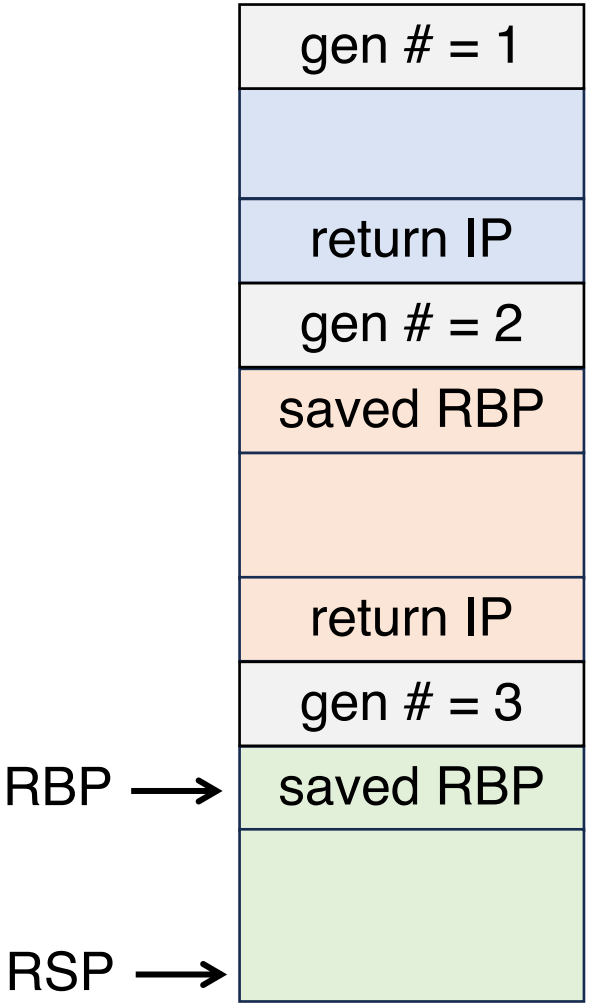
Generation Number

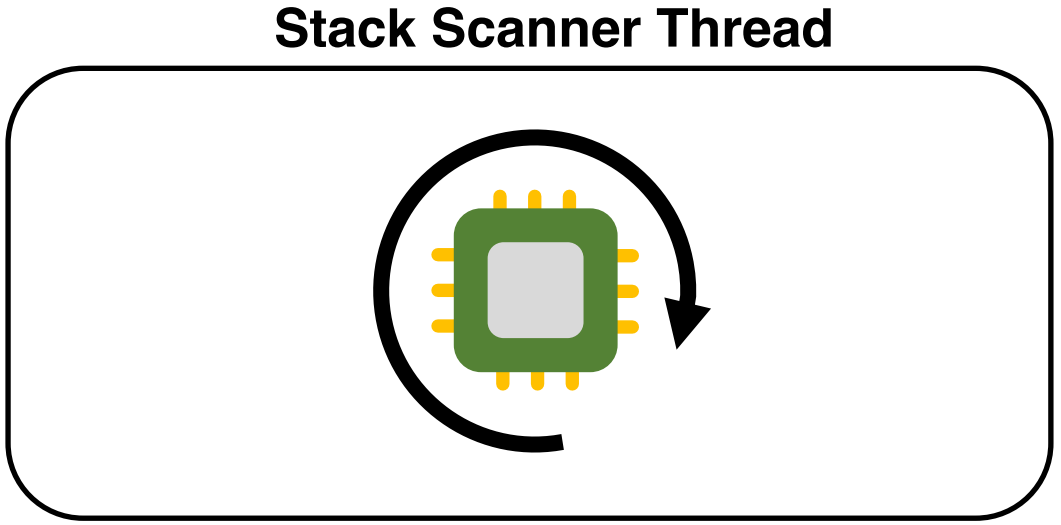


Challenge #3. Cache Thrashing

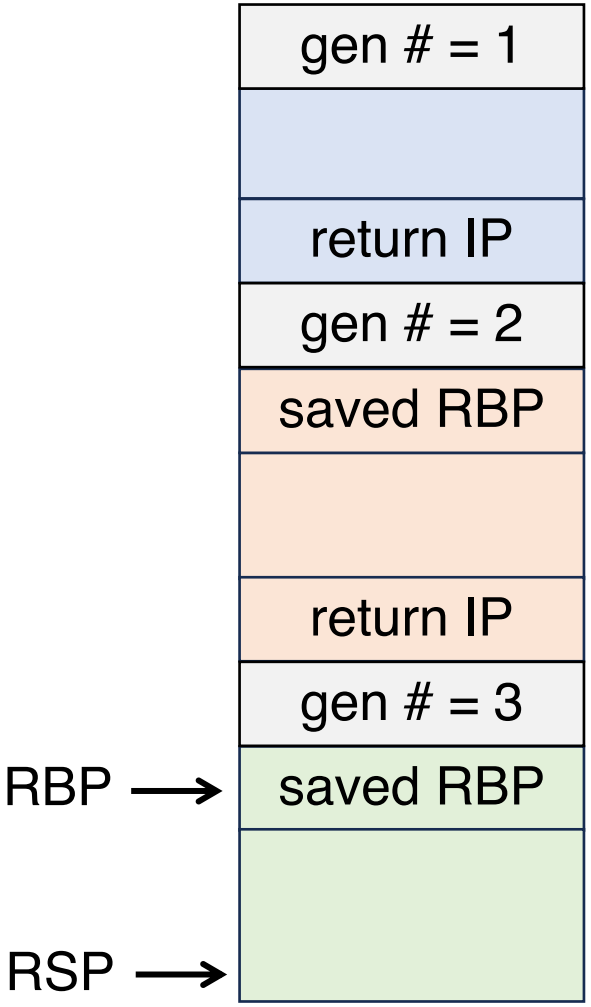


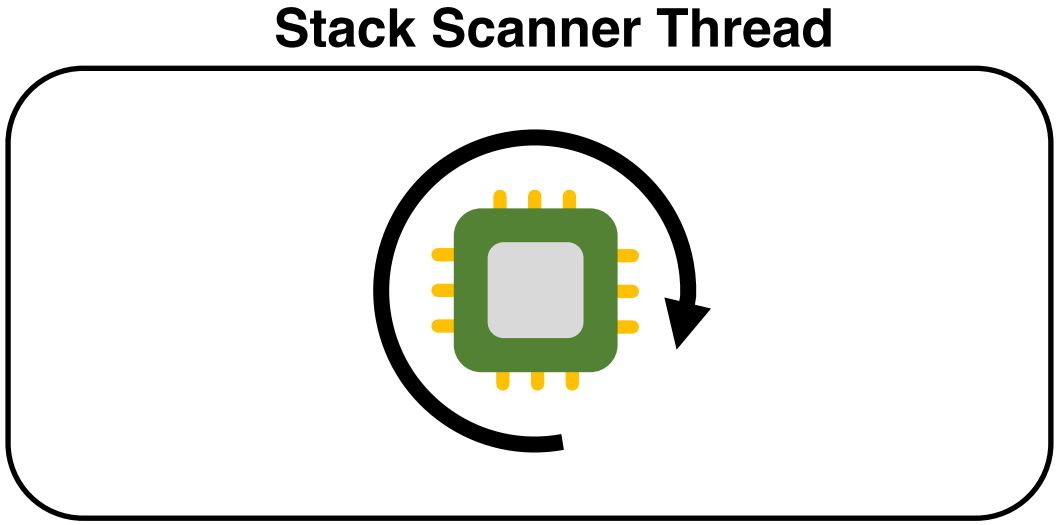
TLS Indicates Stack Frame Change



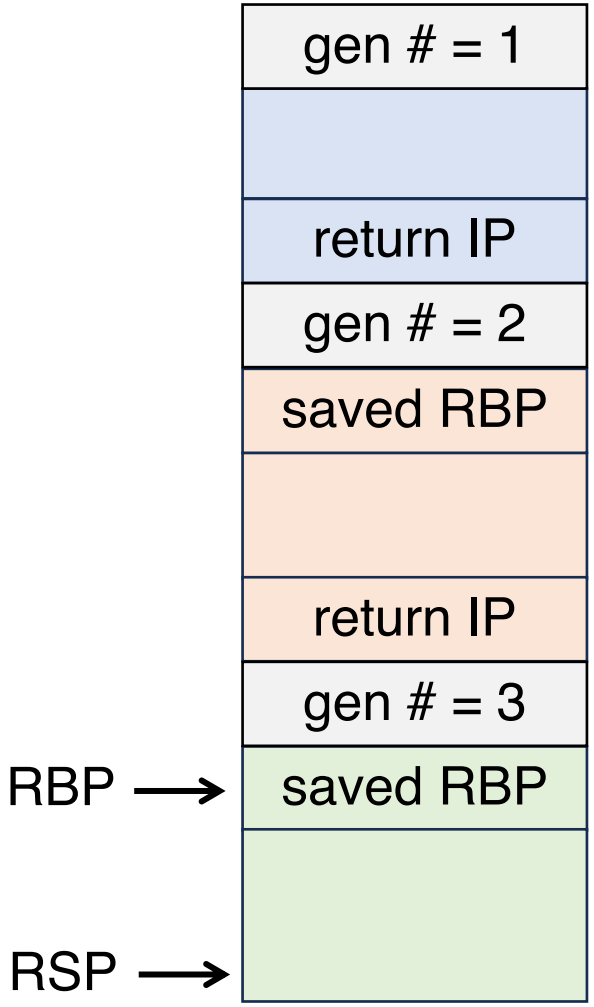


TLS Indicates Stack Frame Change





Latency Measurement



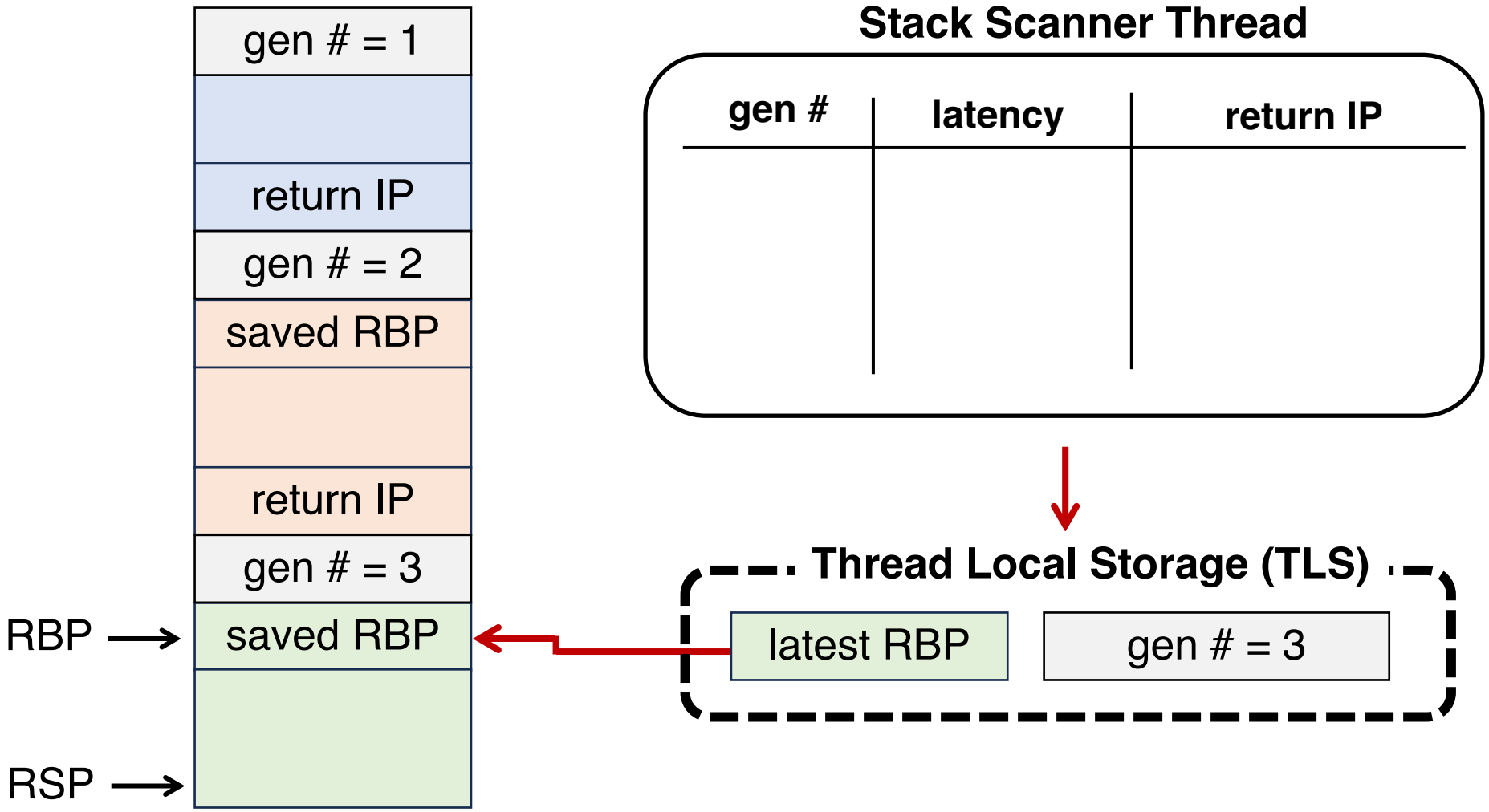
Stack Scanner Thread

gen #	latency	return IP

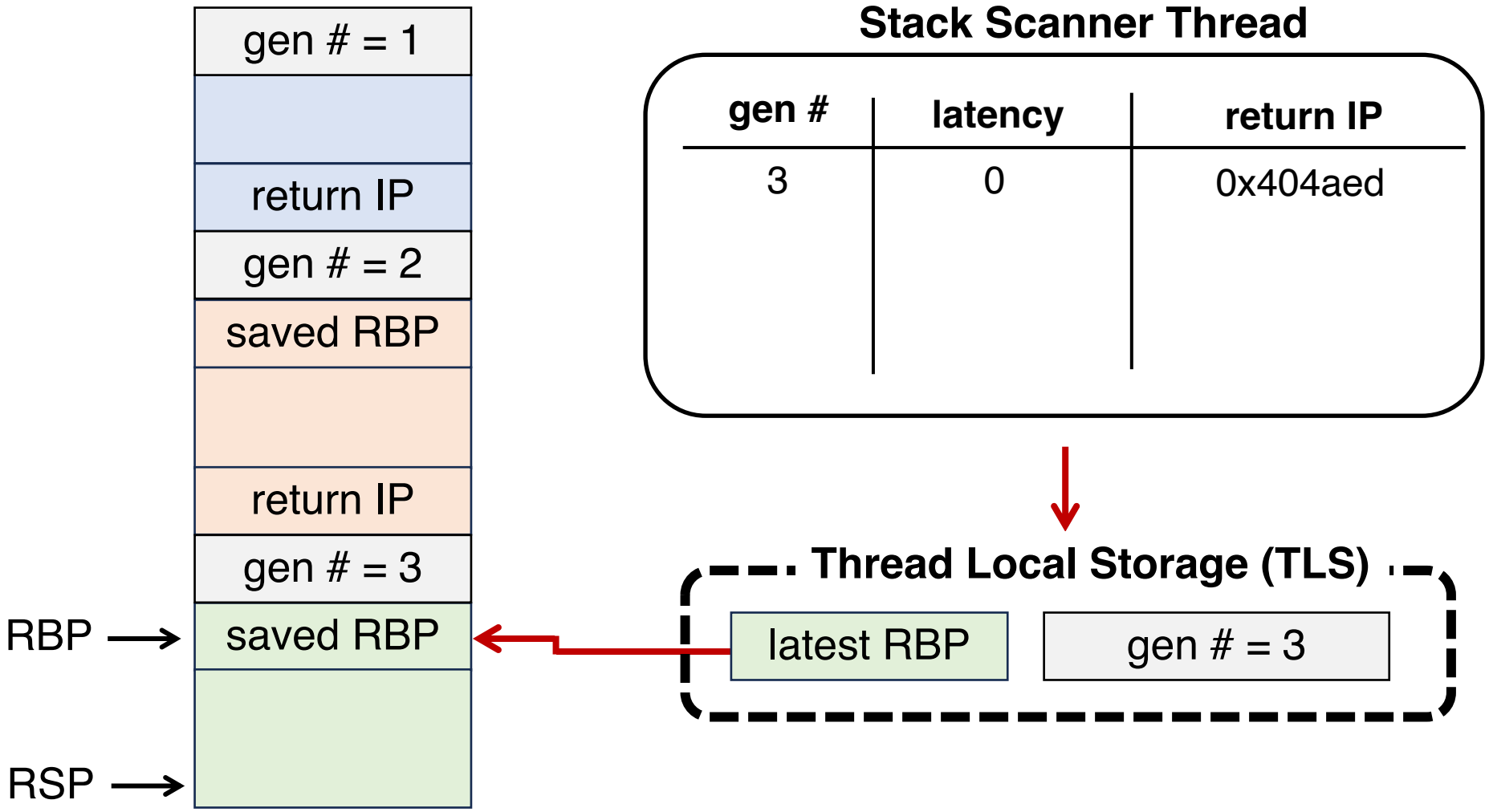
Thread Local Storage (TLS)

latest RBP	gen # = 3

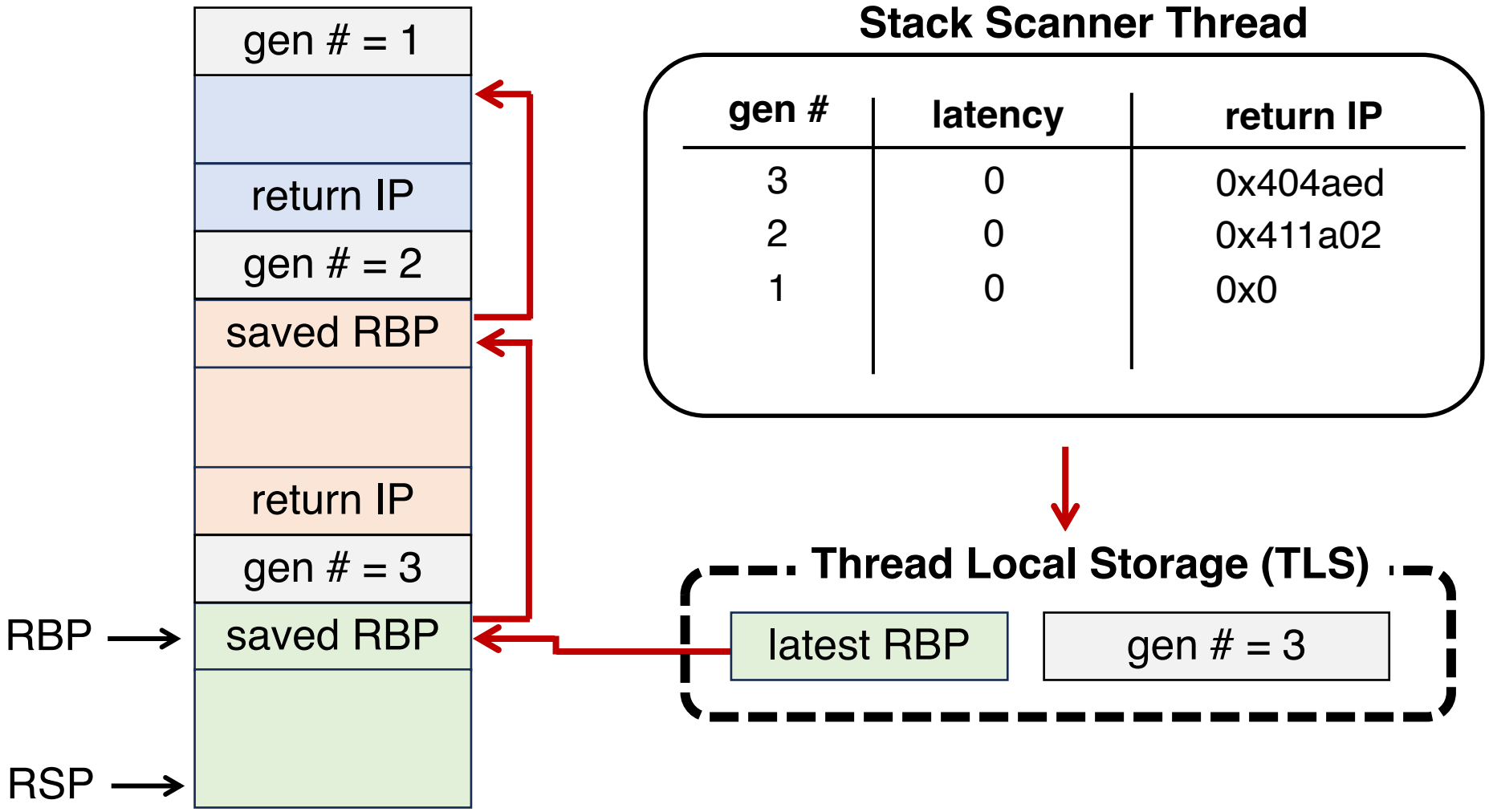
Latency Measurement



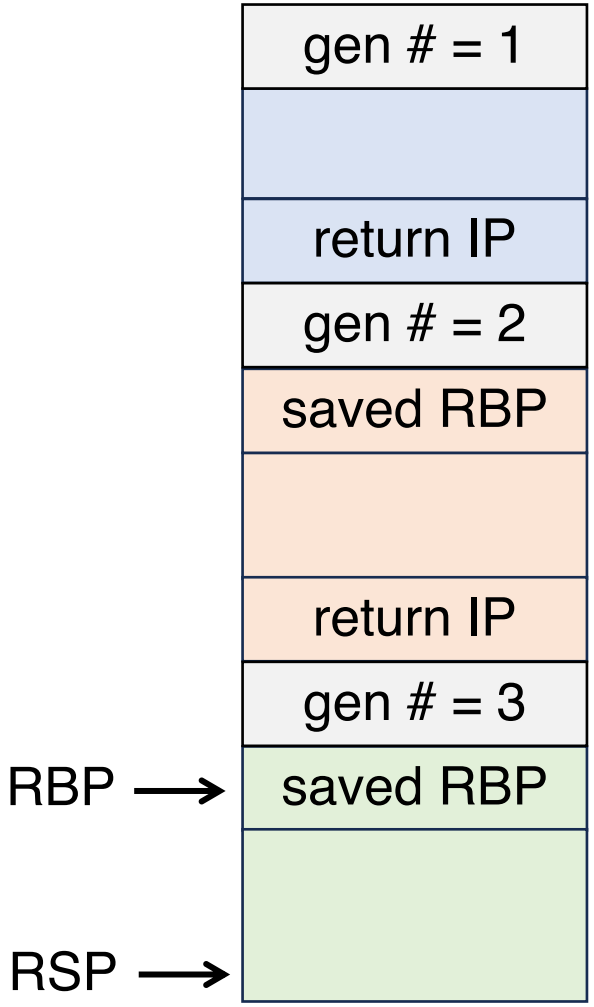
Latency Measurement



Latency Measurement



Latency Measurement



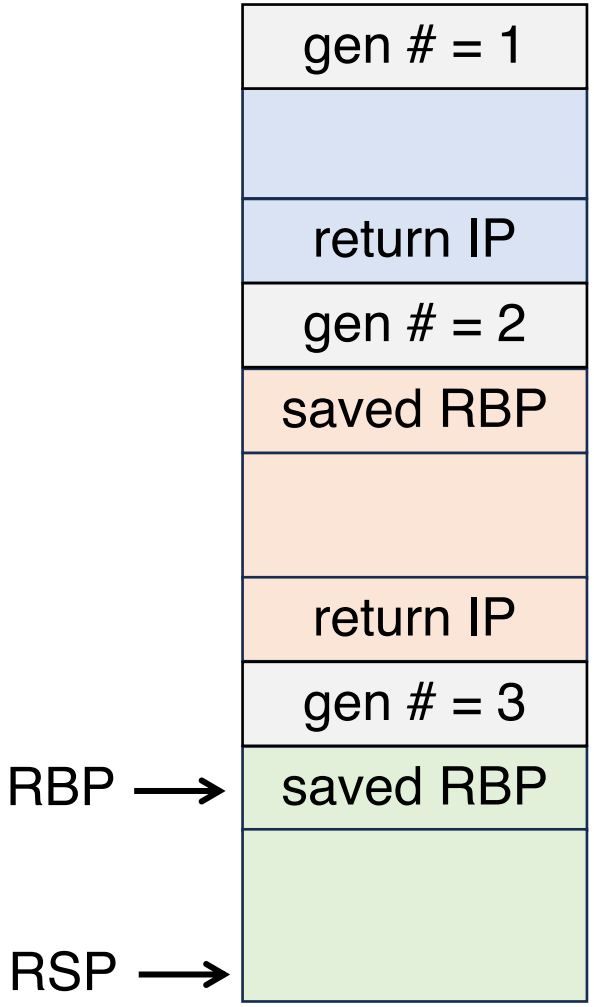
Stack Scanner Thread

gen #	latency	return IP
3	0 + 1	0x404aed
2	0 + 1	0x411a02
1	0 + 1	0x0

Thread Local Storage (TLS)

latest RBP | gen # = 3

Latency Measurement



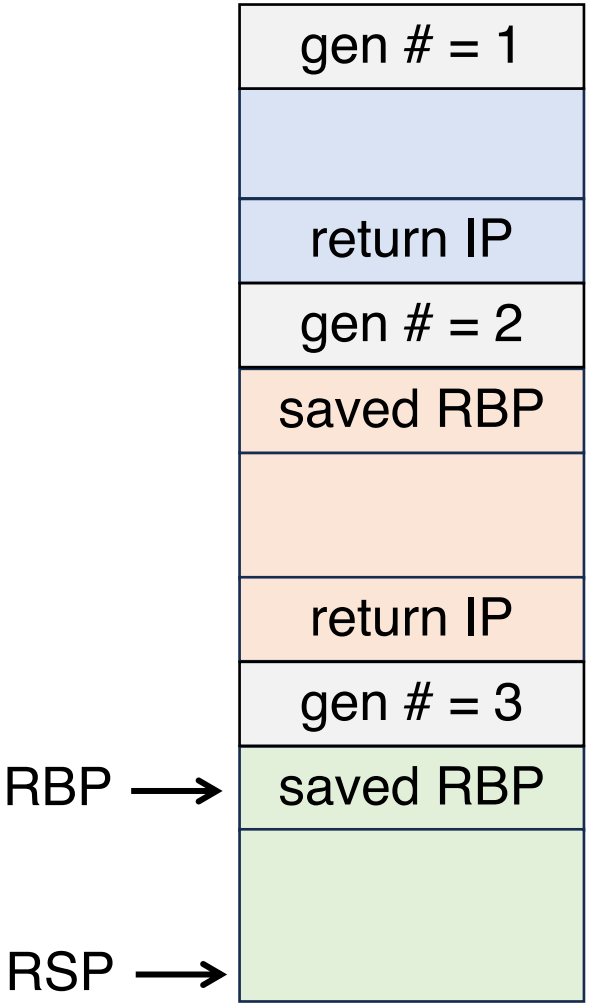
Stack Scanner Thread

gen #	latency	return IP
3	1 + 1	0x404aed
2	1 + 1	0x411a02
1	1 + 1	0x0

Thread Local Storage (TLS)

latest RBP | gen # = 3

Latency Measurement



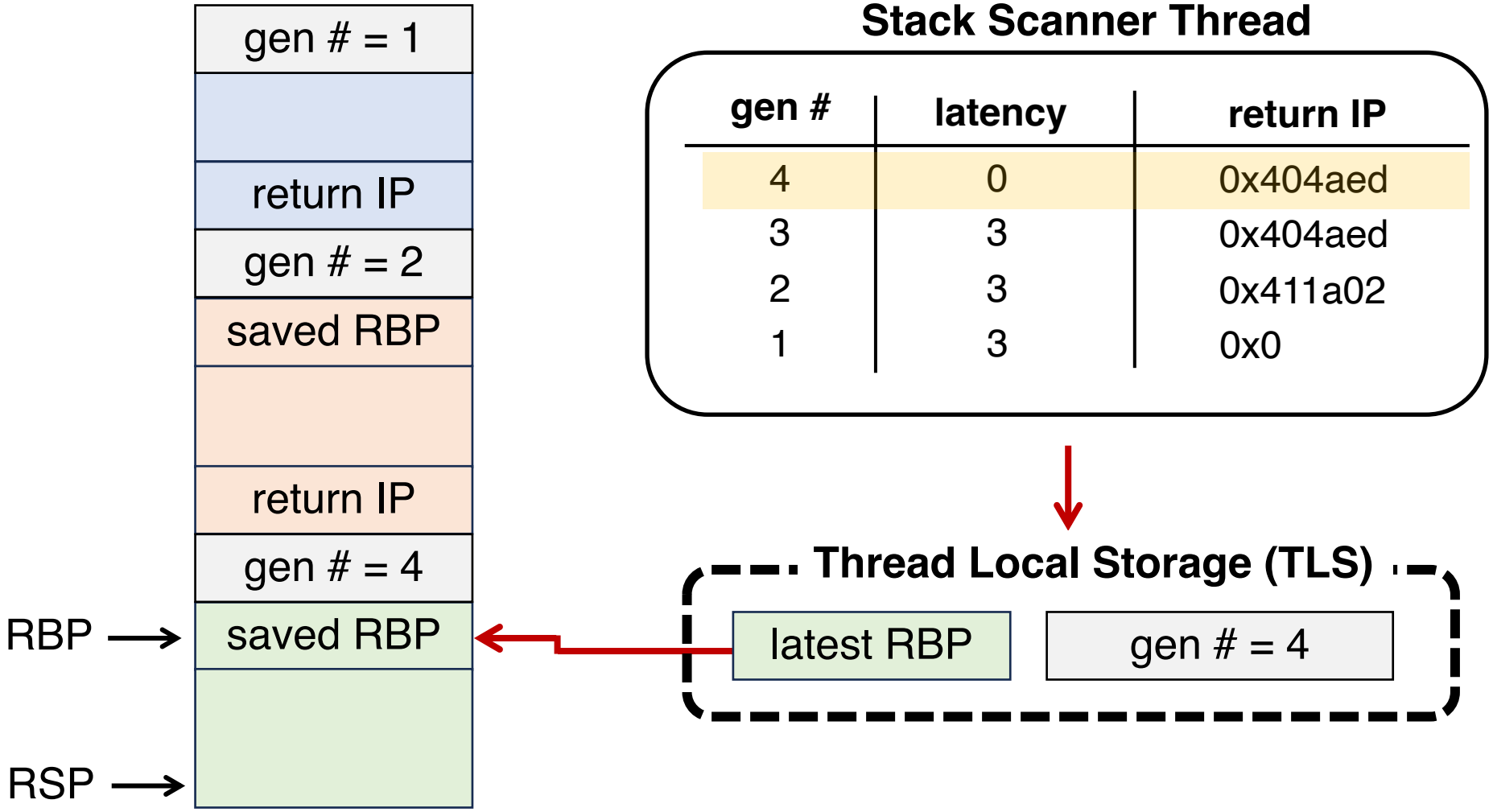
Stack Scanner Thread

gen #	latency	return IP
3	2 + 1	0x404aed
2	2 + 1	0x411a02
1	2 + 1	0x0

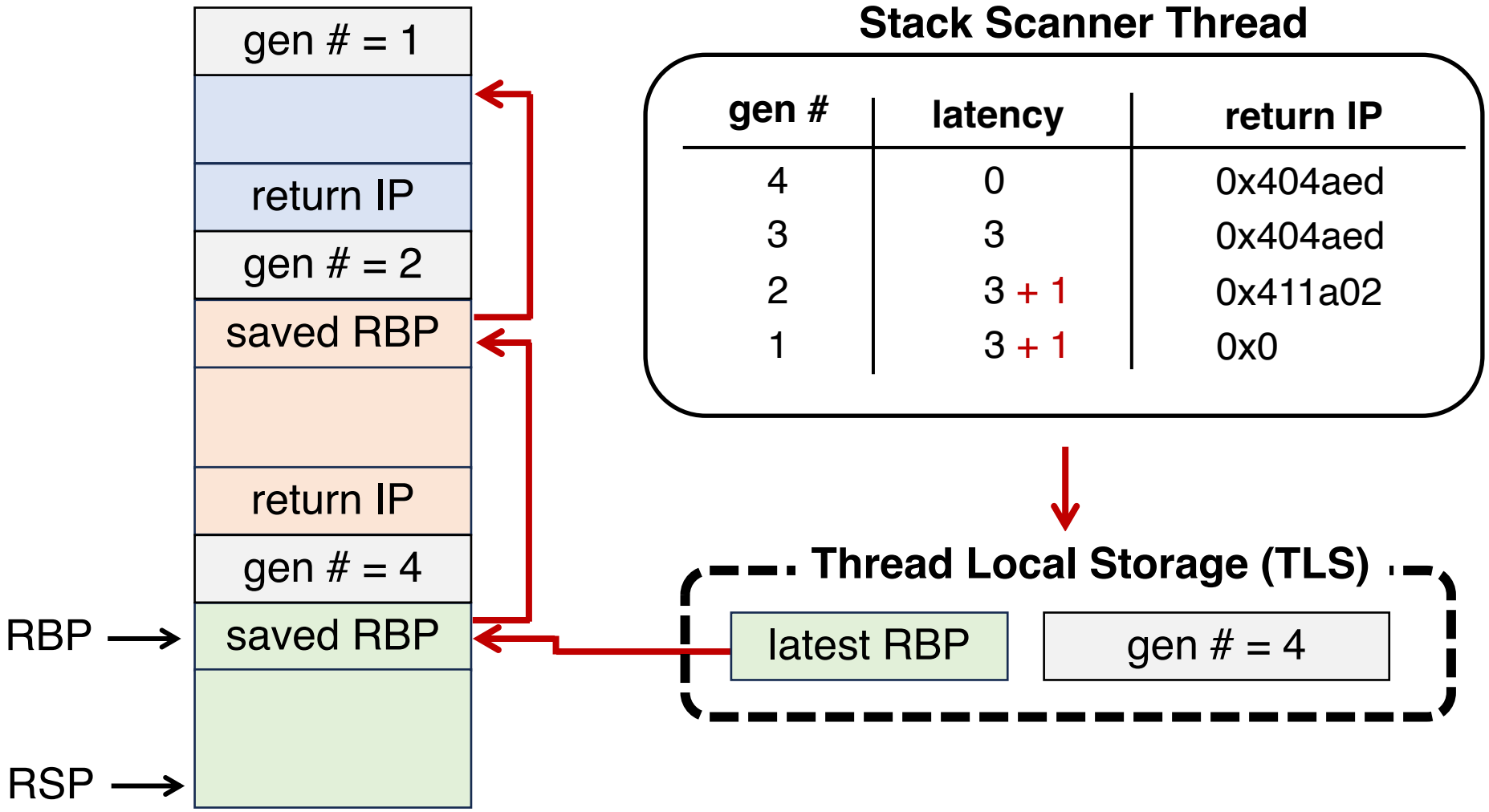
Thread Local Storage (TLS)

latest RBP | gen # = 3

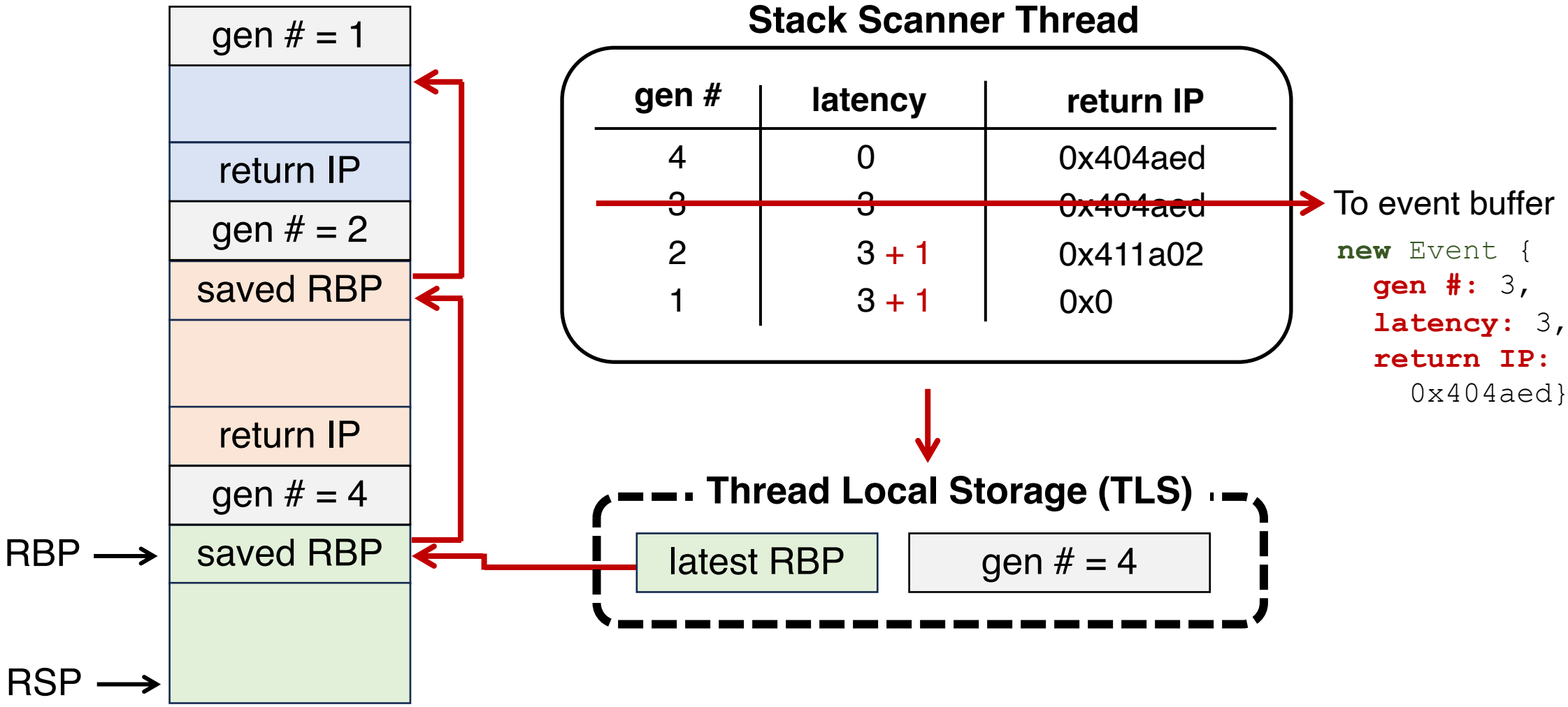
Latency Measurement



Latency Measurement



Latency Measurement



Other Types of Events

Synchronization: mutex, condvar, etc.
- Through shim layer

Threads: thread creation, destroy
- Through shim layer

Requests: request start, request end, request block
- Through optional tagging by developer

Scheduling: context switch, CPU migrate
- From external tool

Evaluation

(1) How much is the overhead of LDB for datacenter applications?
(2) How fast LDB can report the result?
(3) How portable is LDB?

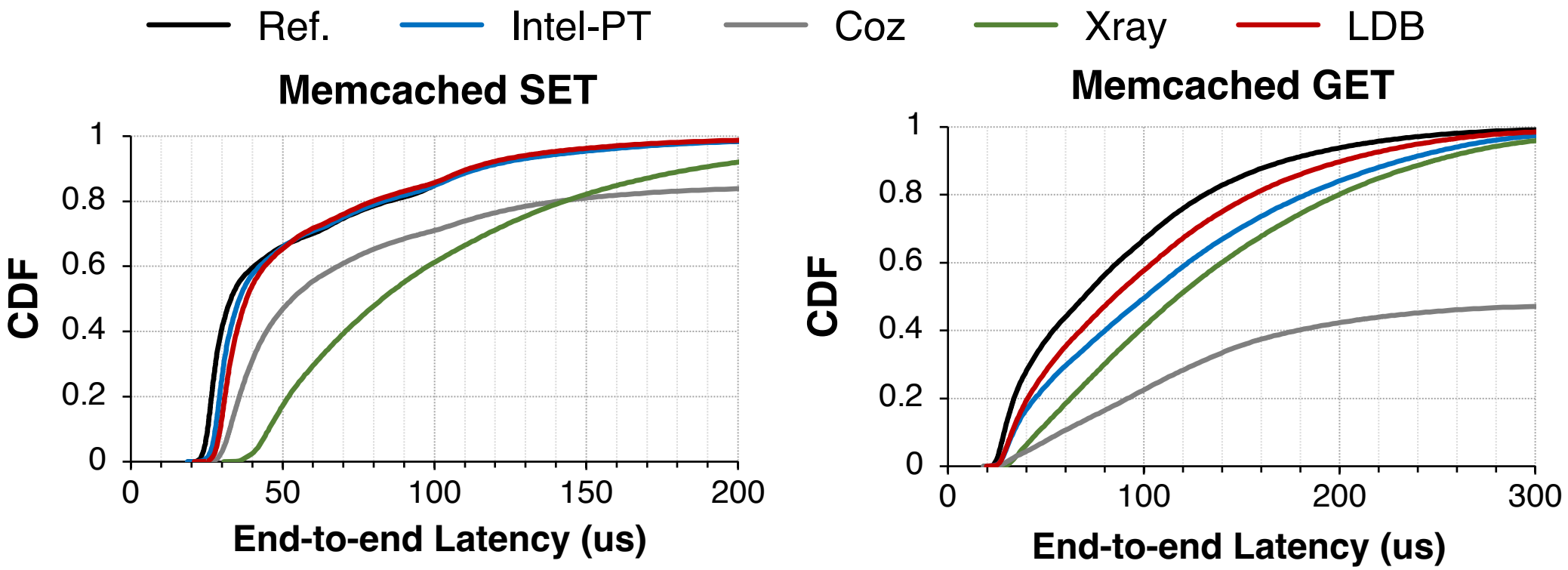
Baselines:

Intel-PT
hardware-assisted processor tracing

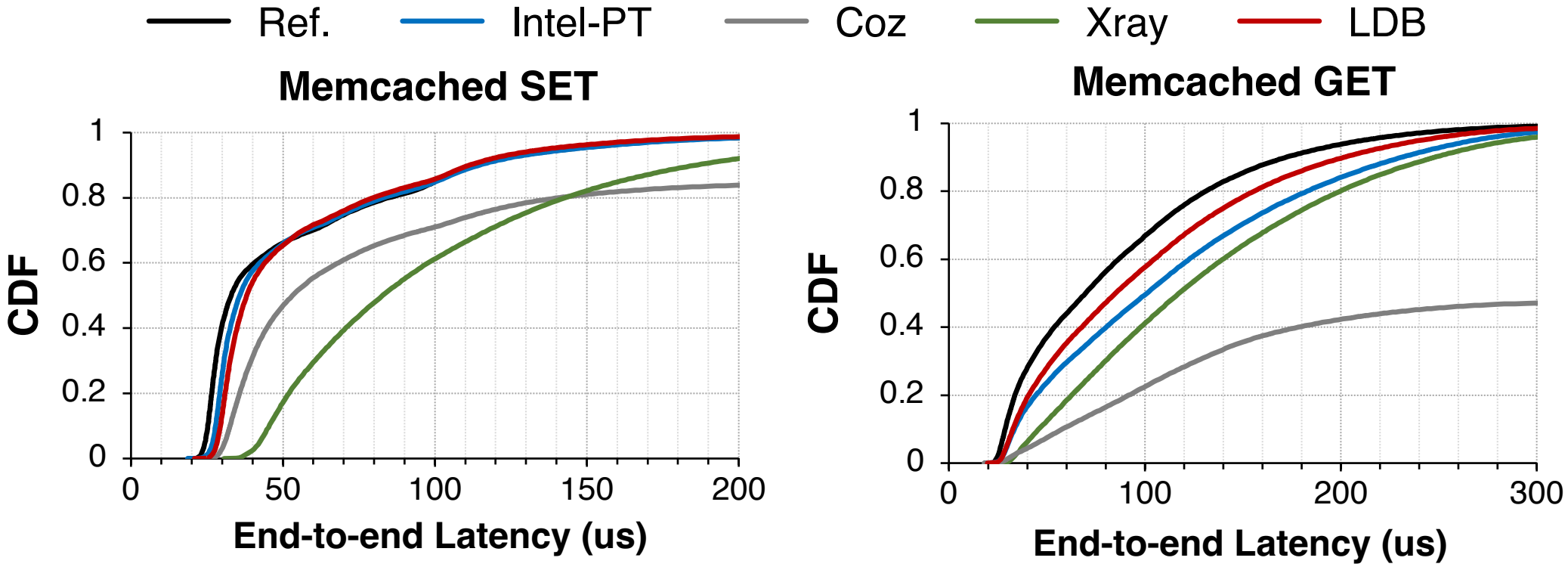
Coz
sampling-based causal latency profiling tool

Xray
timestamp-based function call tracing system with static instrumentation by compiler

Runtime Overhead

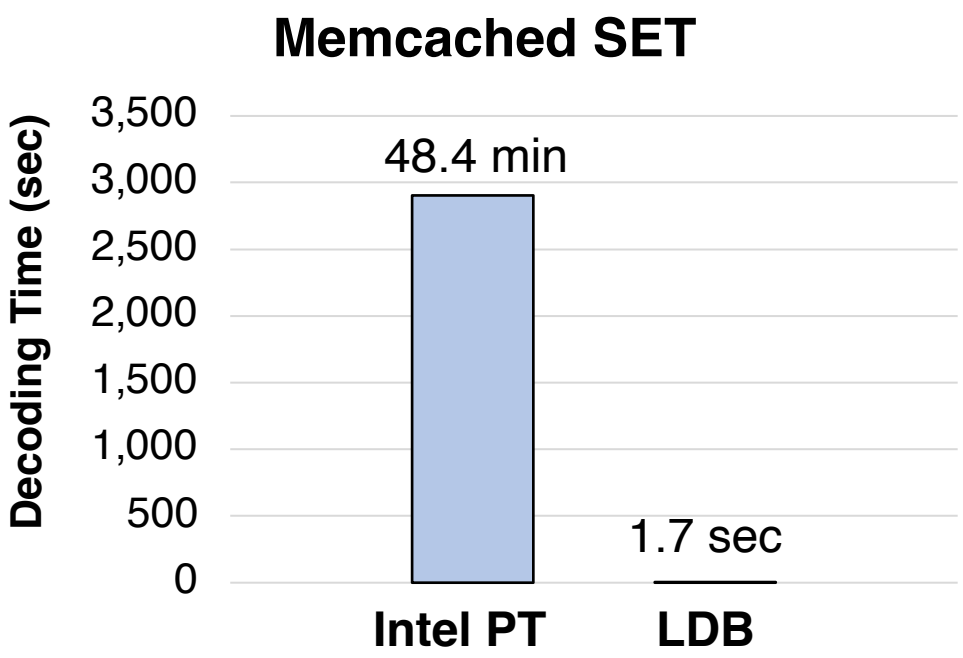


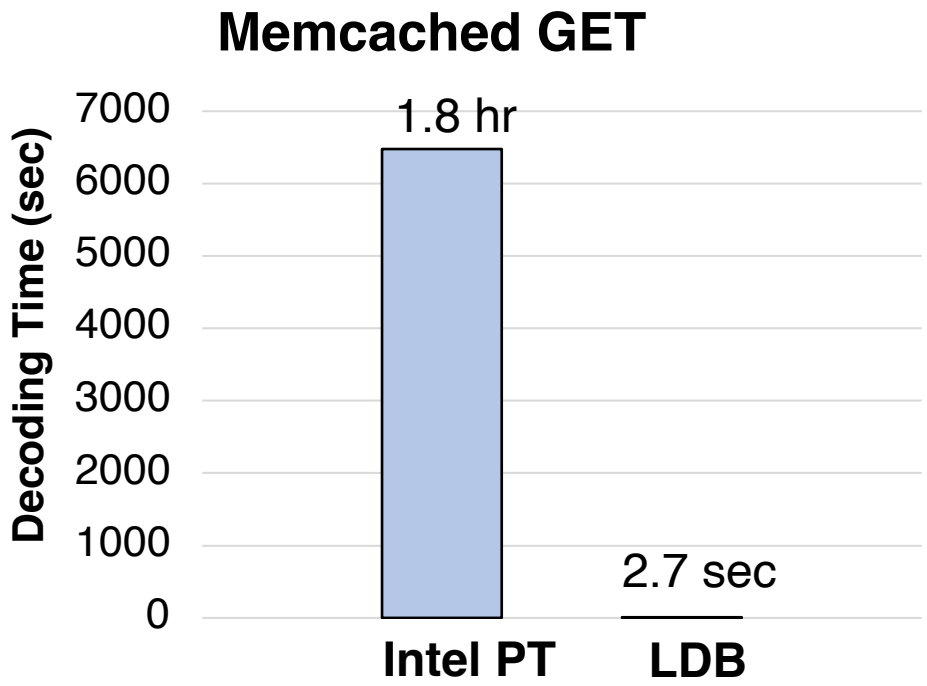
Runtime Overhead



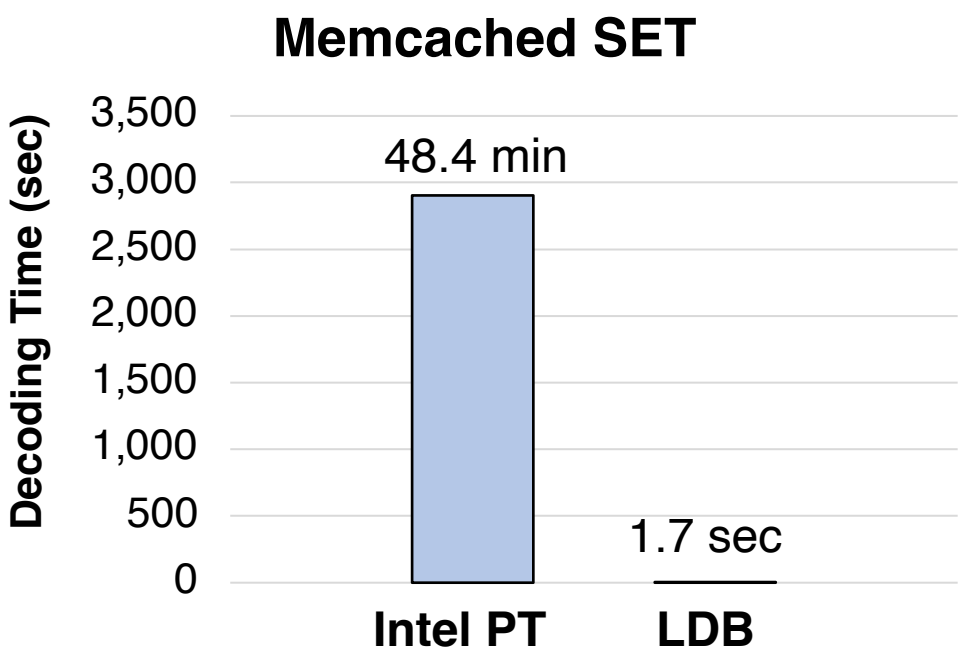
LDB's runtime overhead is comparable to Intel-PT

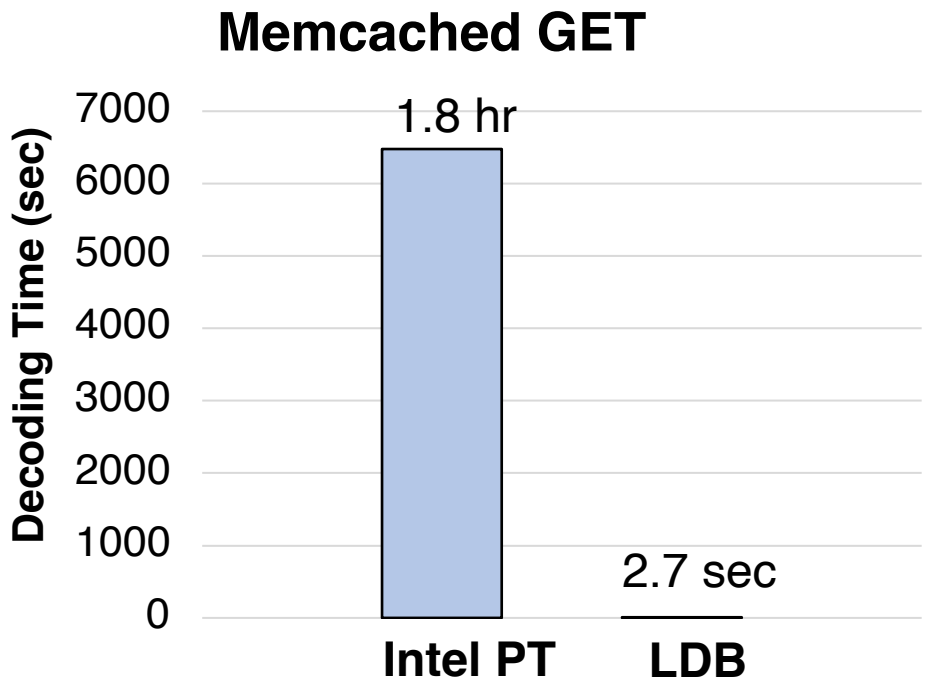
Decoding Time (for 1s trace)





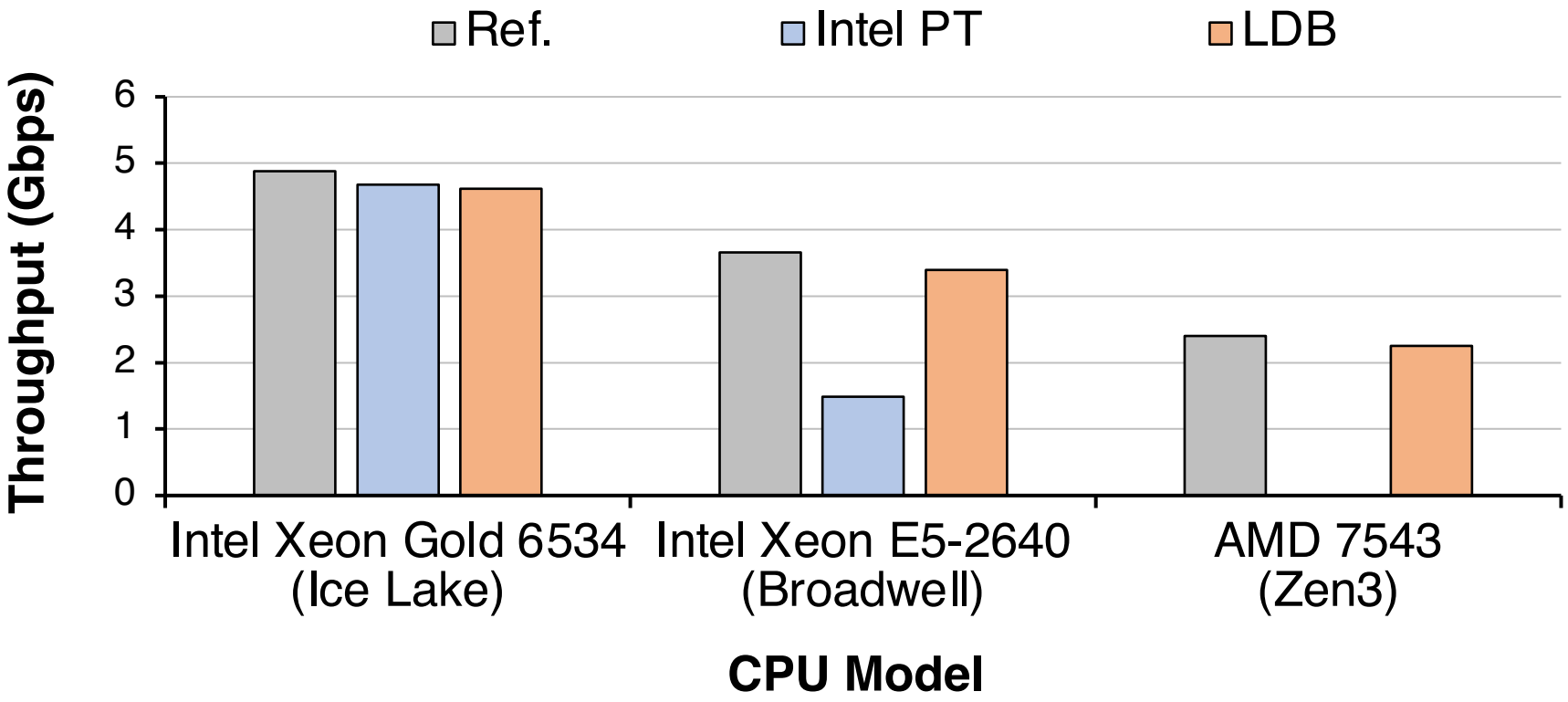
Decoding Time (for 1s trace)



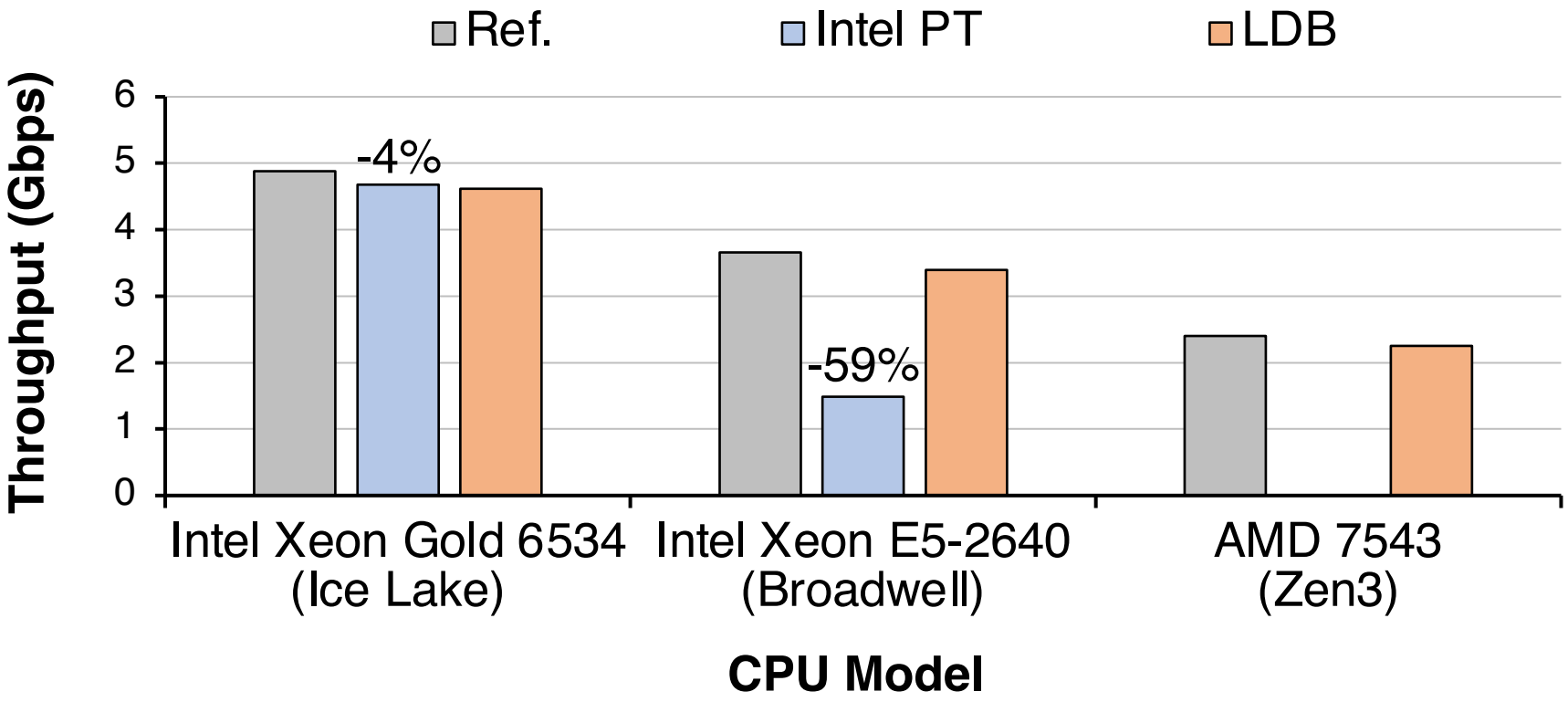


LDB enables interactive debugging with fast results

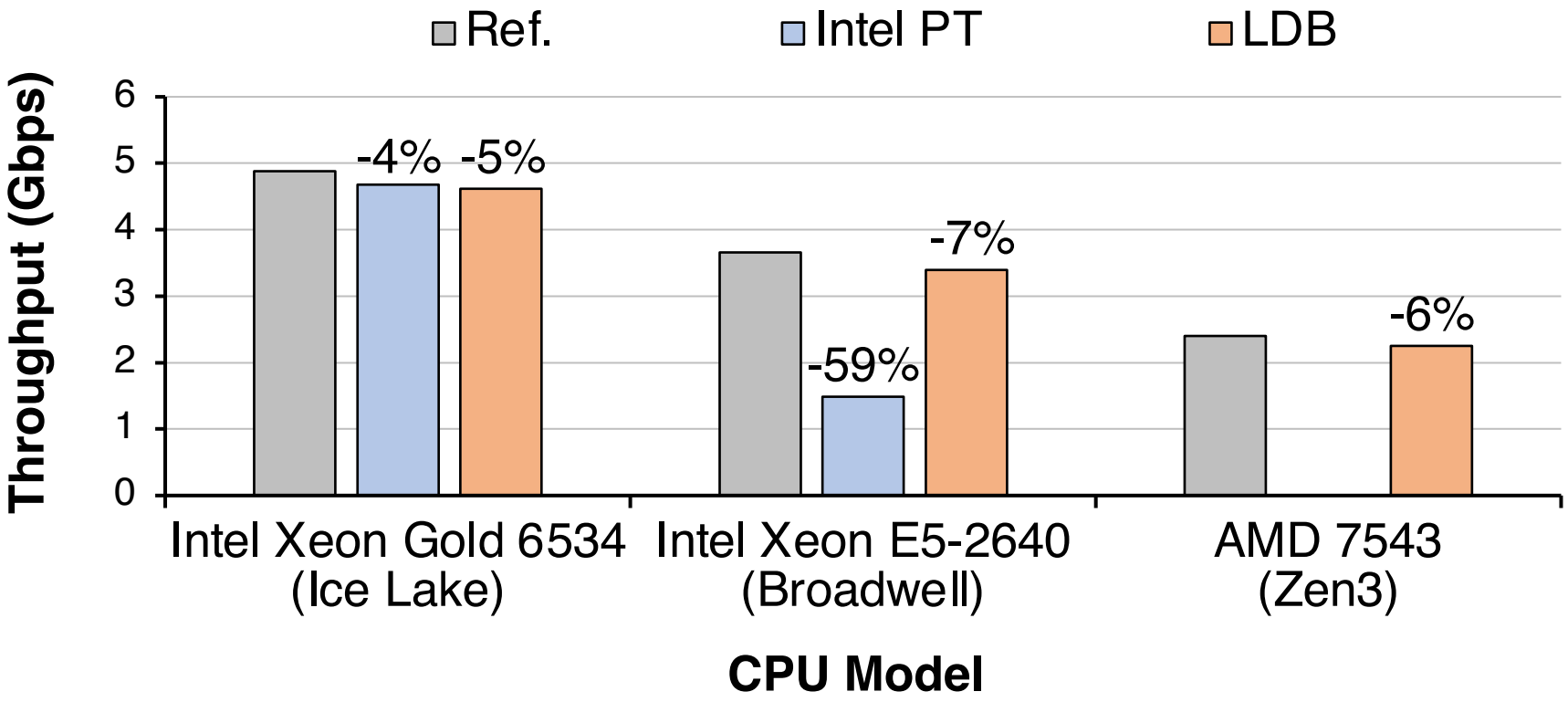
Performance on Different Architectures



Performance on Different Architectures



Performance on Different Architectures



LDB is portable across the different CPU architecture with consistent low overload

Conclusion

- LDB is an efficient **latency profiling** tool to diagnose root causes of **tail latency**.
- LDB's key components include
  (1) Stack sampling with generation number
  (2) Optional request tagging
  (3) Automatic bindings to thread, synchronization events
- Our evaluation shoes that LDB achieves
  (1) Low runtime overhead
  (2) Fast decoding time for interactive debugging
  (3) Great portability

Thank you!

LDB is available at

inhocho89.github.io/ldb/