KubeCon CloudNativeCon

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

AI_dev

Open Source GenAI & ML Summit

China 2024

OpenTelemetry Amplified: Full Observability with EBPF-Enabled Distributed Tracing

Kai Liu[1], Wanqi Yang[2]
[1]Alibaba Cloud, [2]Sun Yat-sen University

Agenda

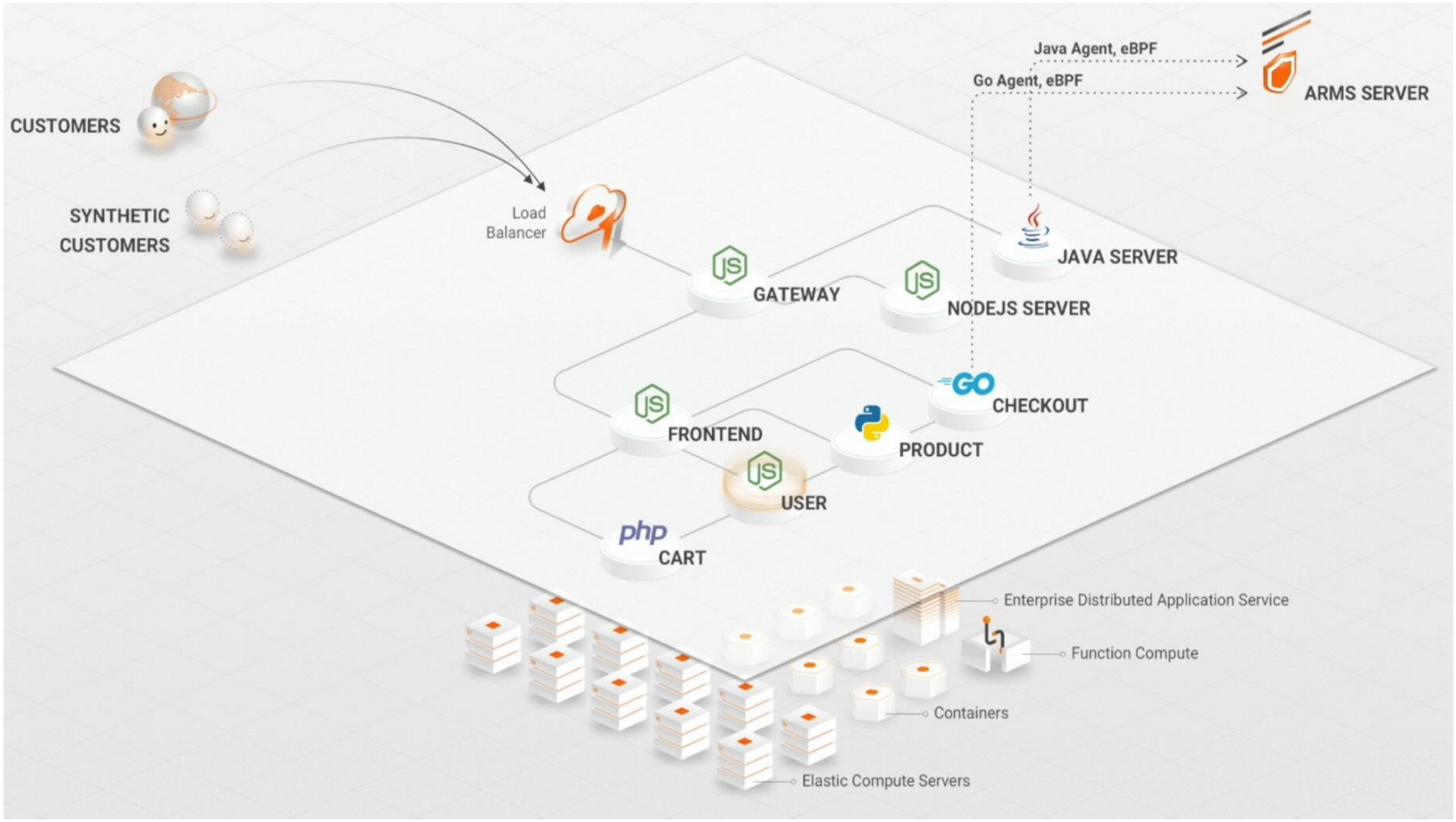
1. Observability in Kubernetes

2. End-to-End Request Tracing

3. Fine-grained Traces

4. Limitations & Future Works

Observability in Kubernetes



Application Observability

- What's the performance of my service?
- Which middleware or services your application depends on?
- What are the dependencies between the various services?

Network Observability

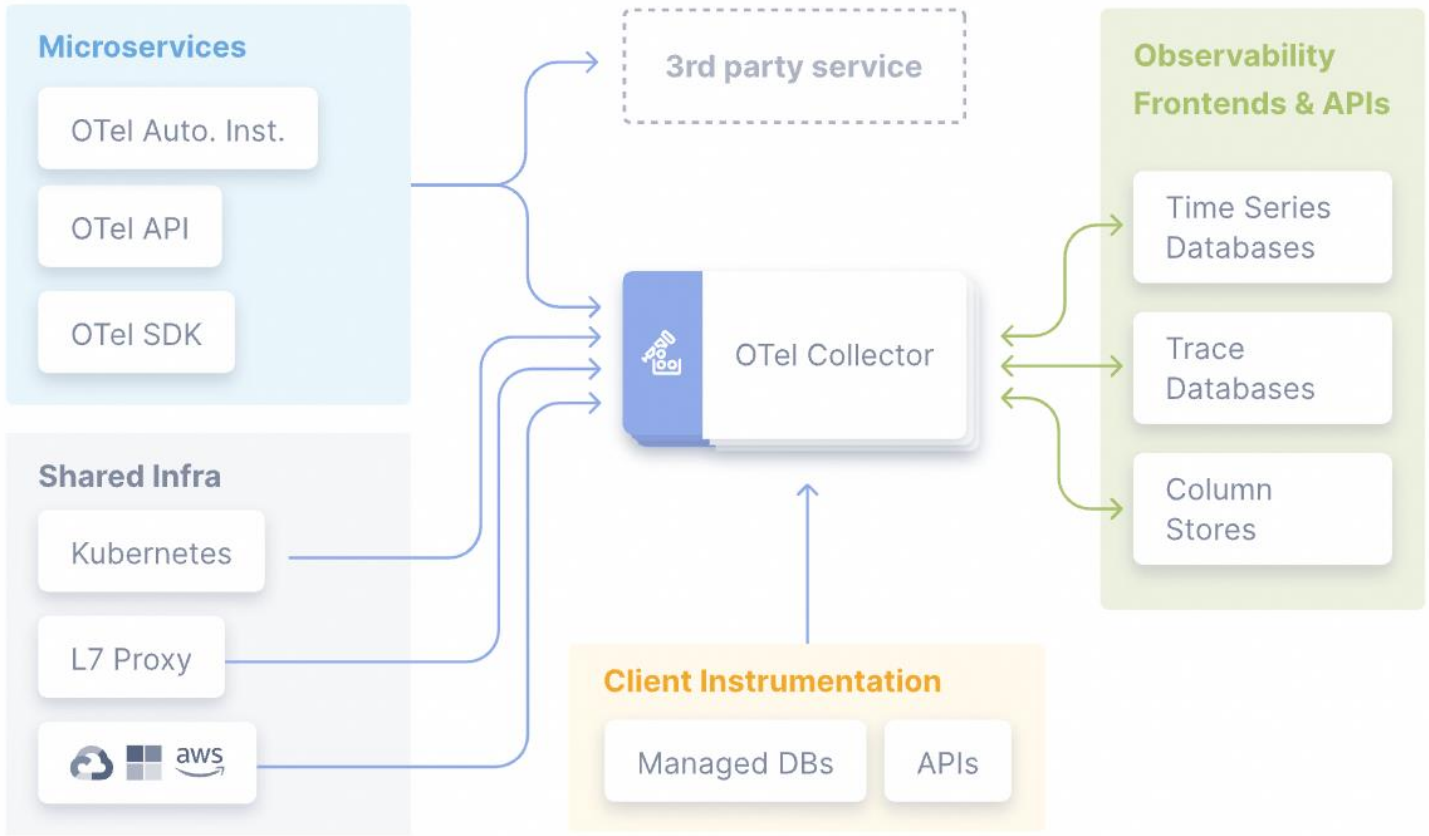
- Is network communication being blocked?
- Is there any problems in the container network?

Security Observability

- What resources has my service operated on?
- What IP addresses has my service requested?

What is Opentelemetry?

- An Observability framework and toolkit



From Opentelemetry.io

Generation Data

- Traces
- Metrics
- Logs

Collection Data

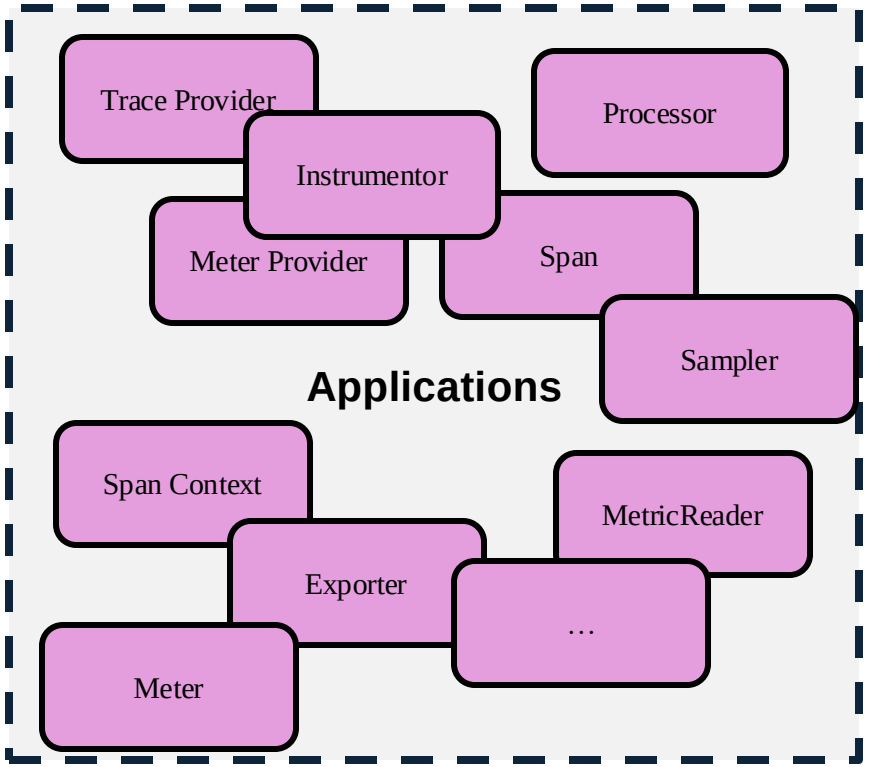
- Convert
- Aggregate
- ...

Management data

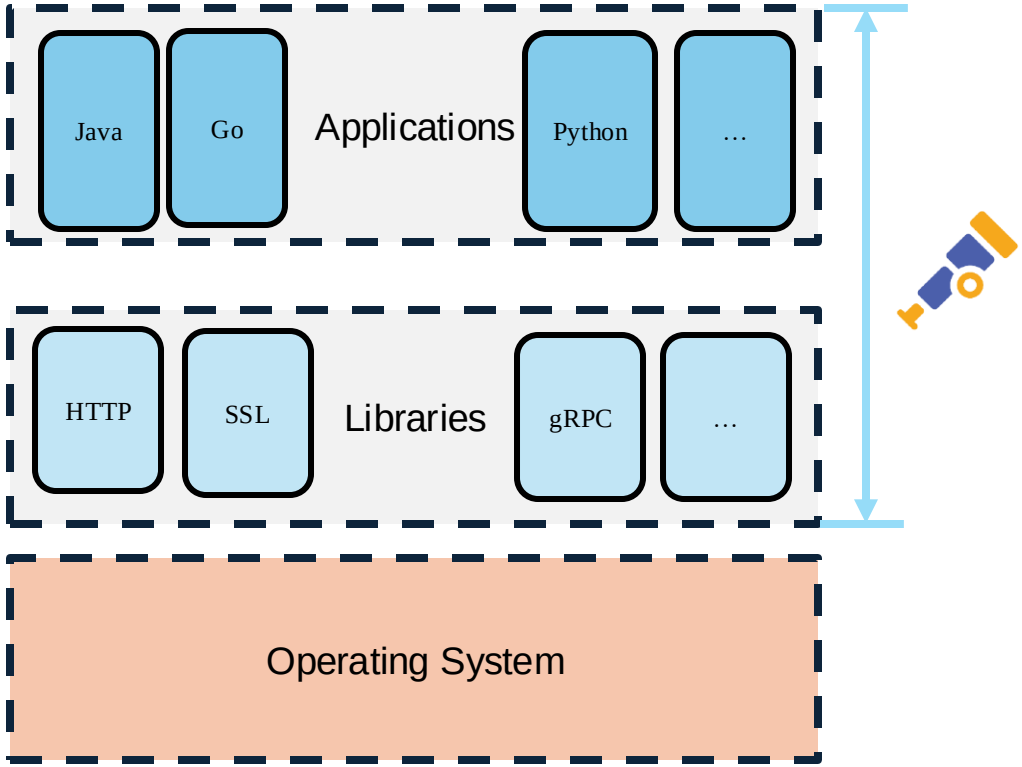
- Grafana
- Prometheus
- Jaeger

Challenges in Opentelemetry

Observability with Opentelemetry is Challenging



Instrumentation



Observability Blind Spots

Agenda

1. Observability in Kubernetes

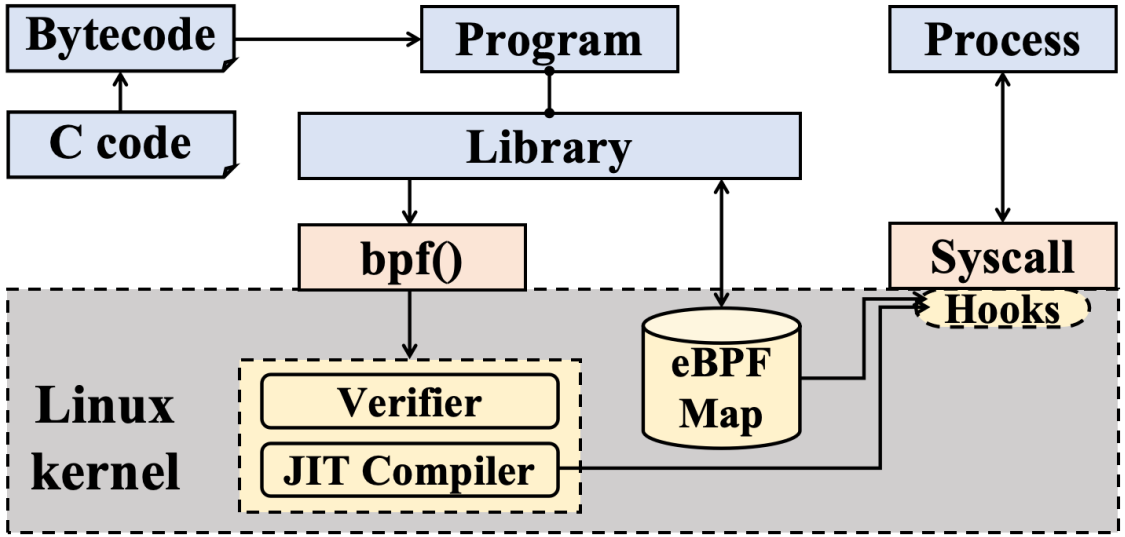
2. End-to-End Request Tracing

3. Fine-grained Traces

4. Limitations & Future Works

What is eBPF ?

- eBPF (extend Berkeley Packet Filter)
  - enables user-defined programs to execute in Linux kernel
  - event-triggered programs attached to specific hooks
    - ✓ network events, file system operations
    - ✓ program types: *kprobe*, *uprobe*, *kretprobe*
  - eBPF Maps enable inter-process communication



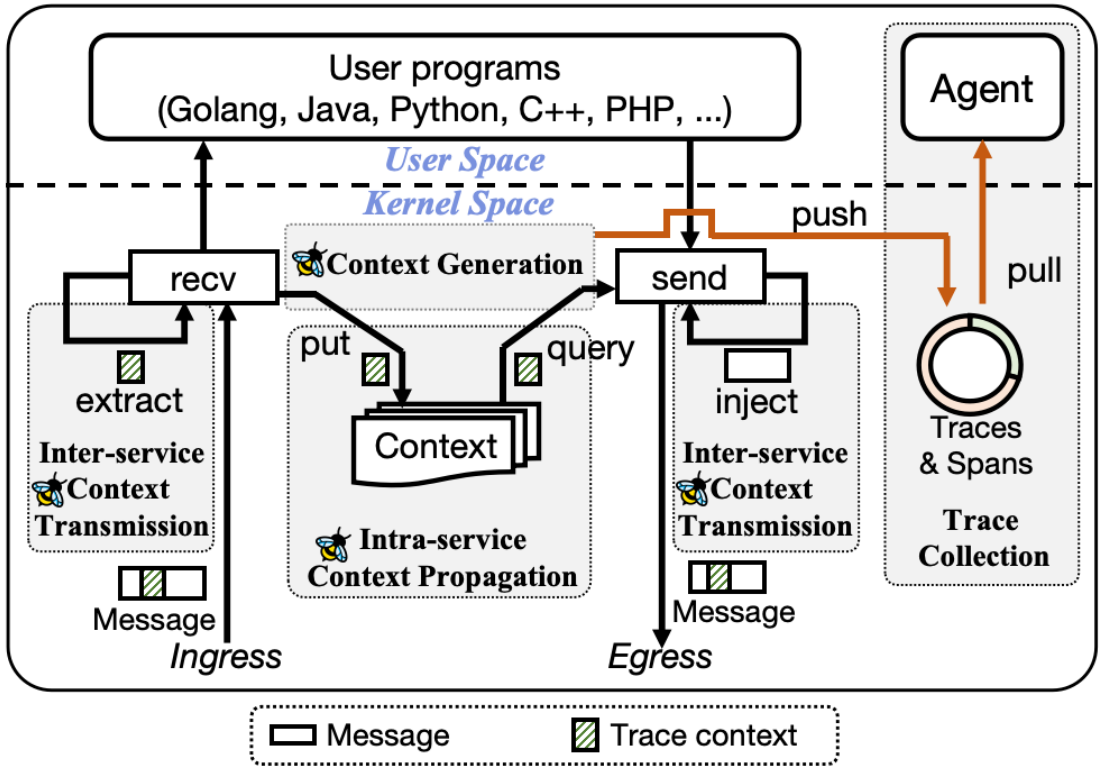
Avantages of eBPF

- in-kernel processing, decoupled from user applications
- no instrumentation into user applications
- compared with Linux kernel module (LKM)
  - high security: security verification
  - pre-defined hooks and program types

	LKM	**eBPF**
Execution environment	Linux	eBPF VM
Degree of security	Low, without security promise	High, with eBPF verifier
Execution time / Overhead	High, less limits on instructions	Low, with strict limits on instructions
Enable in-kernel processing	Yes	Yes

eBPF-enabled Distributed Tracing Systems

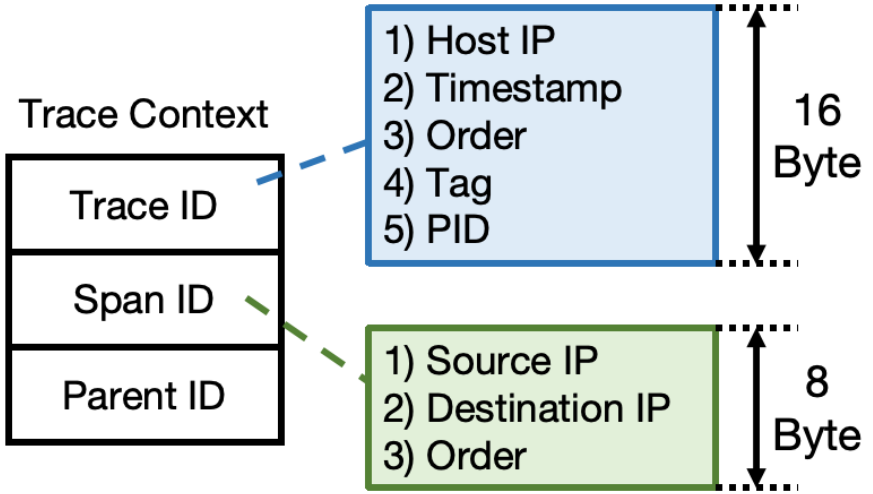
- Existing OTel eBPF-solution
  - support go
  - support limited header keys
- Trace Generation
  - Context Generation
    - ✓ Trace ID, Span ID
  - Inter-service Context Propagation
    - ✓ context transmittion in network
  - Intra-service Context Propagation
    - ✓ request causality
- Trace Collection



OpenTelemetry Go Automatic Instrumentation: https://github.com/open-telemetry/opentelemetry-go-instrumentation

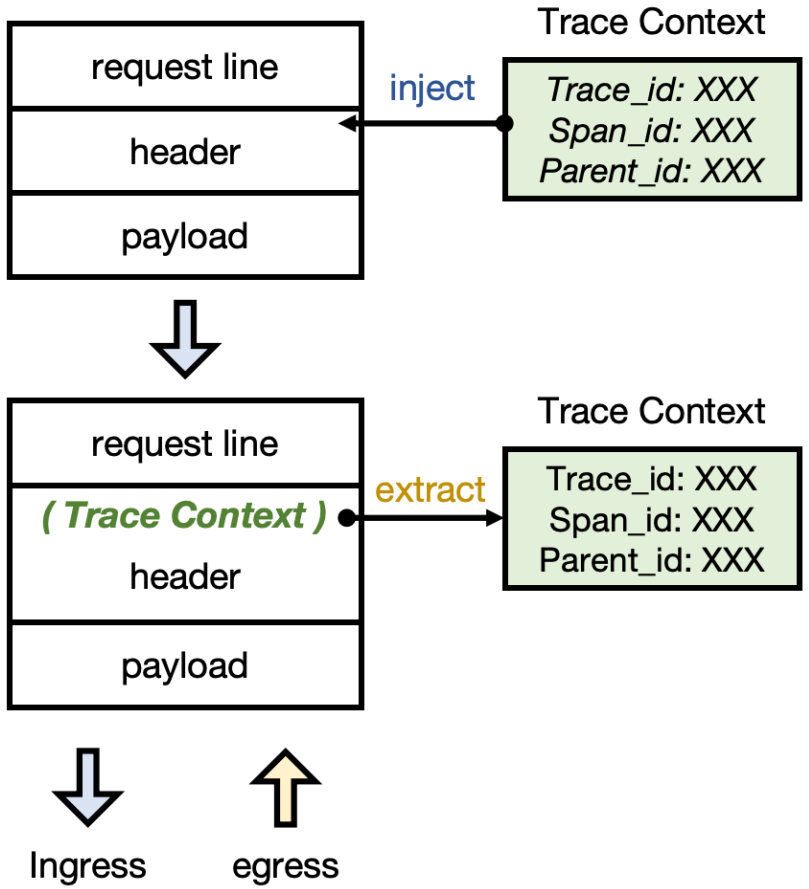
Context Generation

- Trace ID (16-byte)
  - host IP address
  - timestamp
  - sequence number / order
  - debugging tag
  - process ID
- span ID (8-byte)
  - source IP address
  - destination IP address
  - sequence number / order



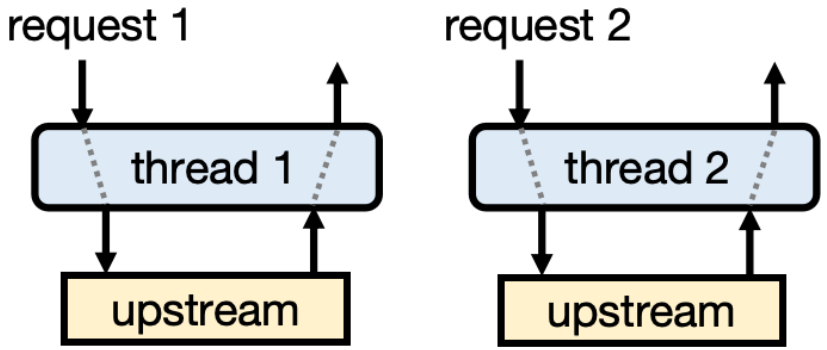
Inter-service Context Propagation

- *propagation trace context through network*
- when receiving messages
  - intercept the **receiving** procedure
    - ✓ *kprobe* on *sock_recvmsg*
  - extract trace context from headers
- when sending messages
  - intercept the **sending** procedure
    - ✓ *sk_msg*
  - inject key-value trace context



Intra-service Context Propagation

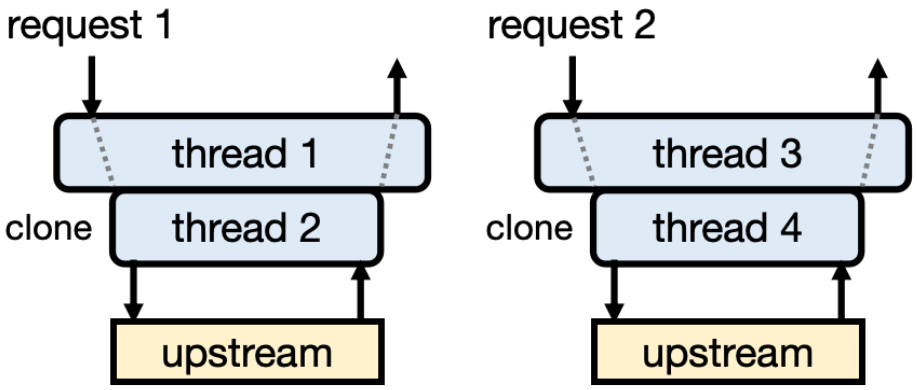
- *progation trace contexts through a service*
  - recognize request causality
  - analyze the execution models of services
    - ***single-threaded*** application
      - ✓ serve a request in a single thread
      - ✓ send the request to the upstream and wait for the response



(a) A single-threaded application.

Intra-service Context Propagation

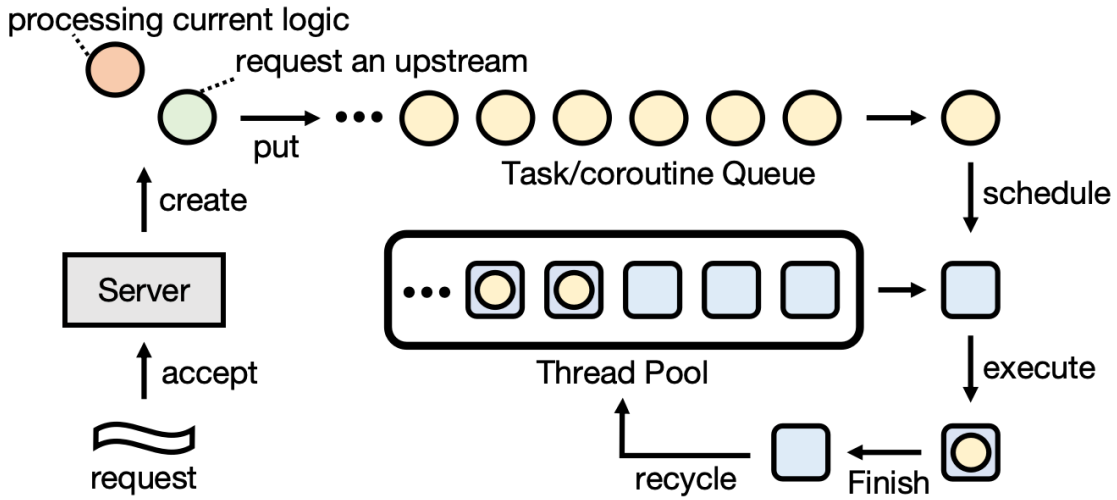
- *progation trace contexts through a service*
  - analyze the execution models of services
    - ***simple multi-threaded*** applications
      - ✓ serve requests by multiple **collaborating threads**
      - ✓ create idle threads to request the upstream



(b) A simple multi-threaded application.

Intra-service Context Propagation

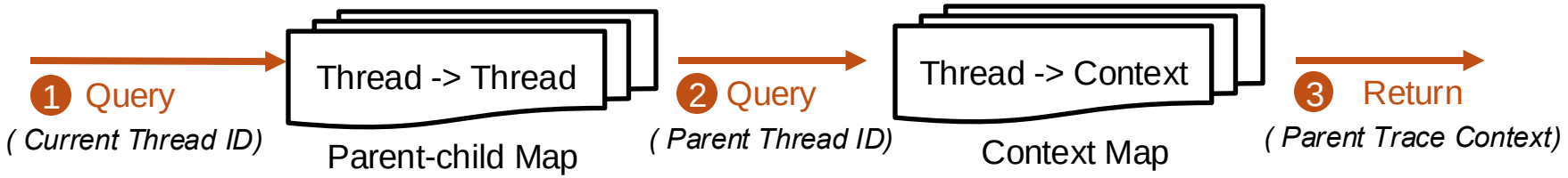
- *progation trace contexts through a service*
  - analyze the execution models of services
    - multi-threaded applications with ***thread pools and coroutines***
      - ✓ serve a request by multiple collaborating tasks or coroutines
      - ✓ create tasks or coroutines to request the upstream and to do other operations



(c) A multi-threaded application in a simple thread pool or coroutines.

Intra-service Context Propagation

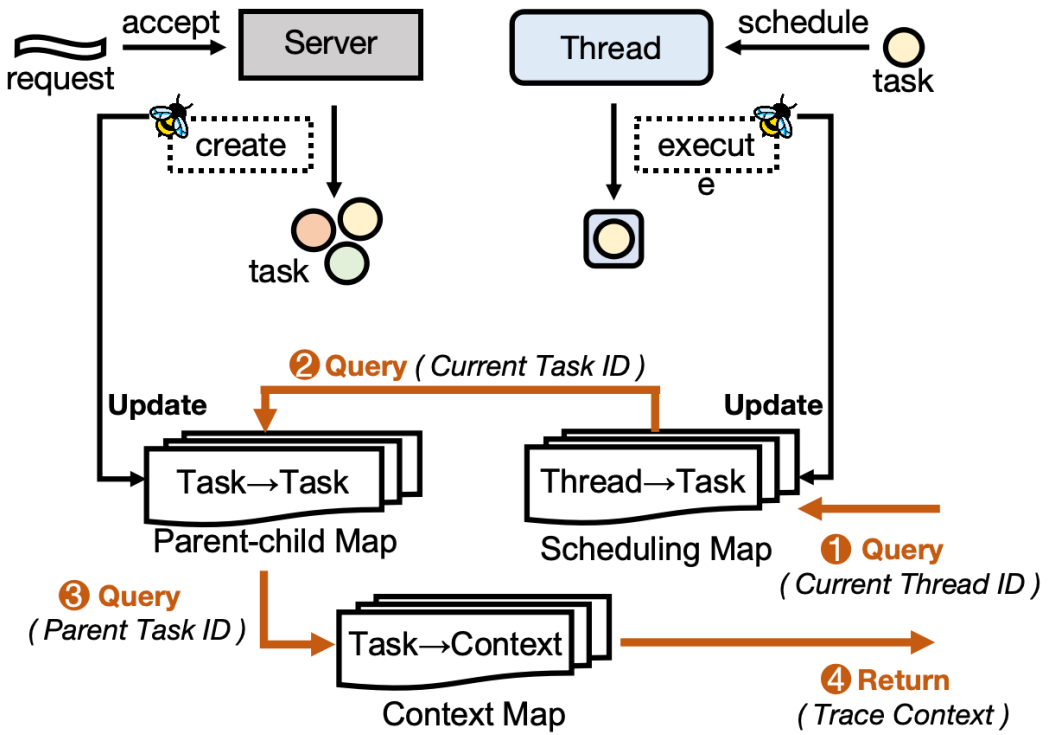
- single-threaded and simple multi-threaded services
  - capture information
    - ✓ thread parent-child relationship
  - when propagating context
    - ✓ find current thread ID
    - ✓ query parent thread ID (for multi-threaded ones)
    - ✓ query parent trace context
    - ✓ generate child trace context



Intra-service Context Propagation

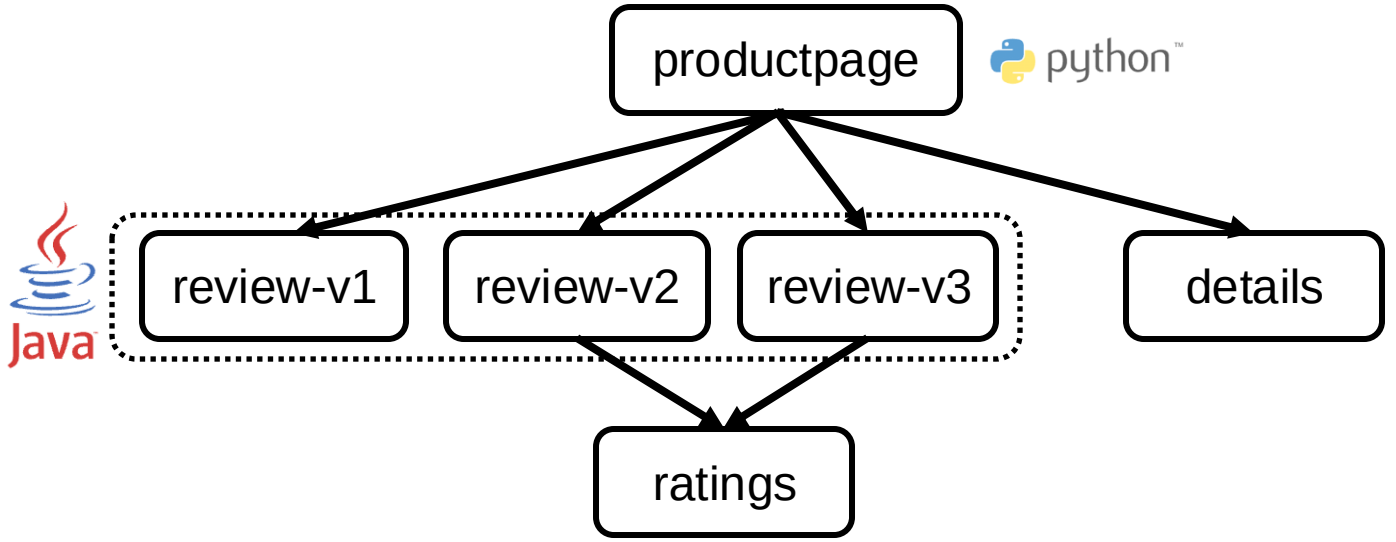
- multi-threaded service in thread pools or coroutine
  - capture information
    - ✓ task parent-child relationship
    - ✓ task scheduling on threads
  - when propagating context
    - ✓ find current task ID
    - ✓ query parent task ID
    - ✓ query parent trace context
    - ✓ generate child trace context

Type	Hook	Functionality
uprobe	execute	acquire coroutine scheduling info
	newproc1	find parent-child relationships of corountines
uretprobe	newproc1	find parent-child relationships of corountines
kpretprobe	alloc_pid	find parent-child relationships of threads



Example Demo

- Bookinfo
  - 4 services



Span ID	Parent ID	Span Name	Timeline (ms)
019e381400000000	0000000000000000	server:productpage_pod->NONE	10.427ms
019e1e2a02000000	019e381400000000	client:productpage_pod->detail_svc	0.894ms
019d019e03000000	019e1e2a02000000	server:detail_pod->productpage_pod	0.585ms
019ec11d05000000	019e381400000000	client:productpage_pod->review_svc	0.66ms
022f019e01000000	019ec11d05000000	server:review1_pod->productpage_pod	0.321ms

System Overhead

- Environment
  - Bookinfo
  - 5 virtual machine
  - 600 concurrent user requests
- Overhead
  - service performance: ~0.5%
  - resouce consumption:
    - ✓ 2.07% CPU Usage
    - ✓ 737 MB Memory

Metric	w/o our system	w/ our system	Overhead (Δ)
Latency (ms)	4593.33	4617.33	24 (0.5%)
QPS	130.14	129.6	0.54 (0.4%)
Memory Usage (%)	62.52	67.01	4.49 (737MB)
CPU Usage (%)	10.14	10.35	0.21 (2.07%)

Agenda

1. Observability in Kubernetes

2. End-to-End Request Tracing

3. Fine-grained Traces

4. Limitations & Future Works

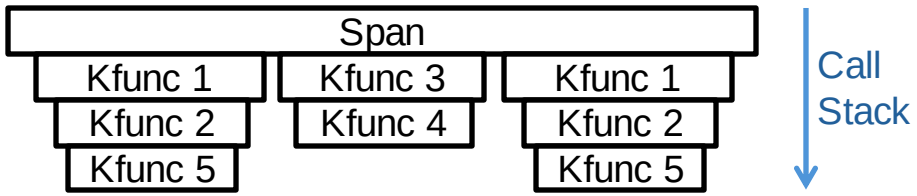
Why we need kernel tracing?

- Distributed tracing systems
  - request-level insights
  - High execution efficiency, low overhead
  - When a a service encounters issues
    - ✓ Are there any errors occurring in the kernel?
    - ✓ Which threads are involved?
    - ✓ Is there a blockage in the network?

Collect critical kernel functions as children spans ?

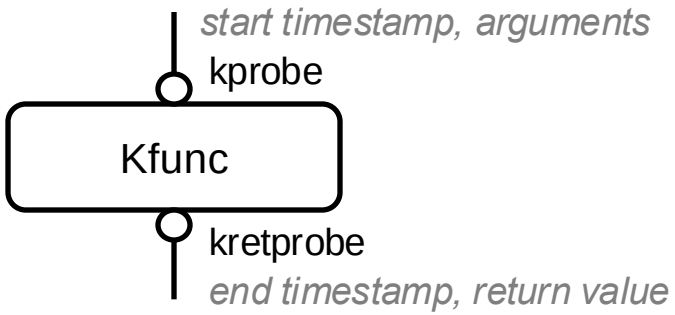
Fine-grained Traces

- Fine-grained Traces
  - Collect telemetry data in Kernel
  - Context Propagation between kernel functions
  - Merge kernel spans with request-level spans



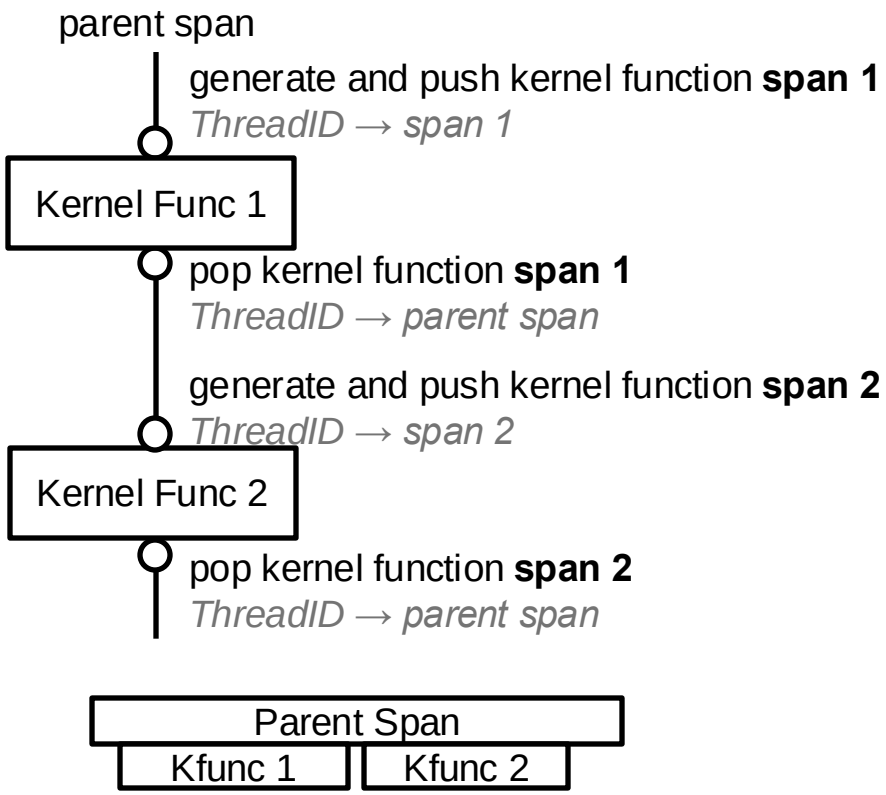
Collect telemetry data in kernel

- Kernel function span generation
  - eBPF hooks
    - ✓ kprobe
    - ✓ kretprobe
  - information
    - ✓ start timestamp
    - ✓ duration
    - ✓ arguments
    - ✓ return value

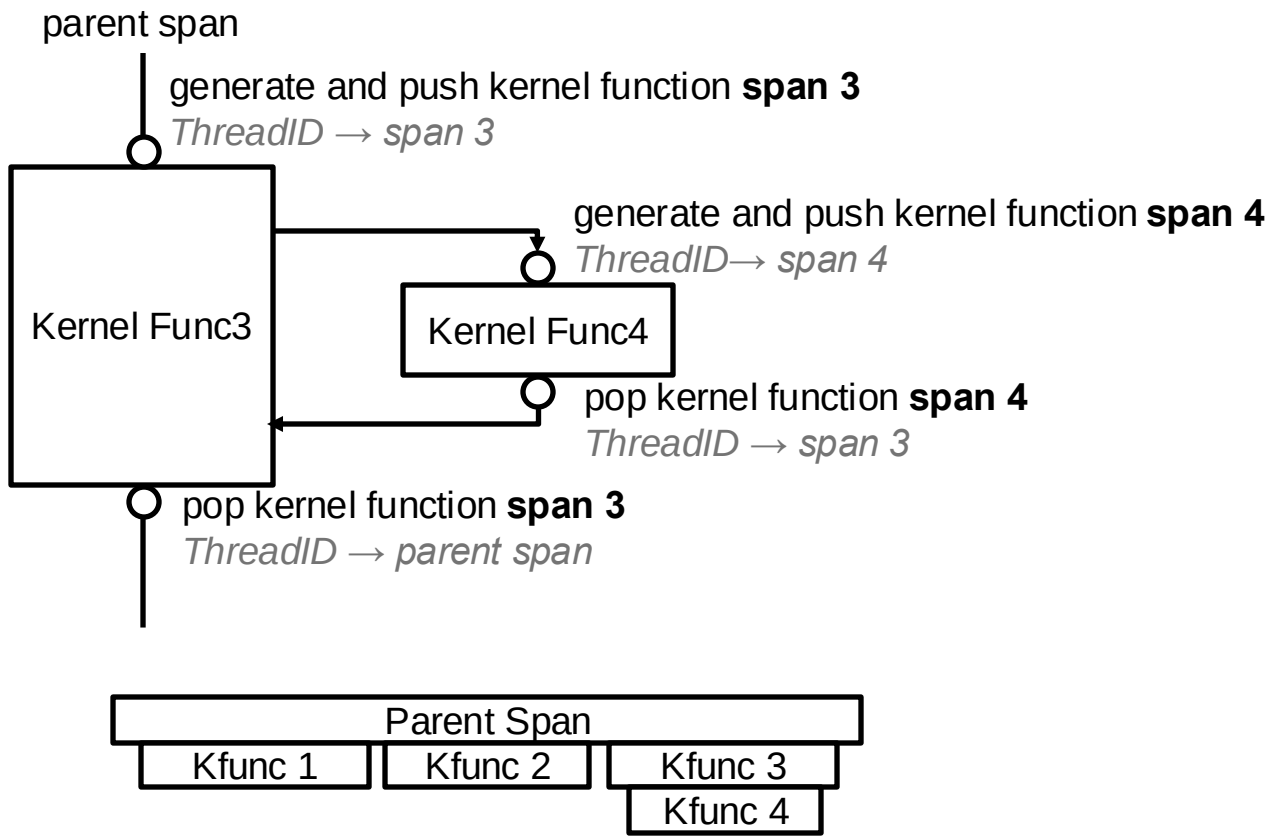


Context Propagation

Synchronous Thread Model



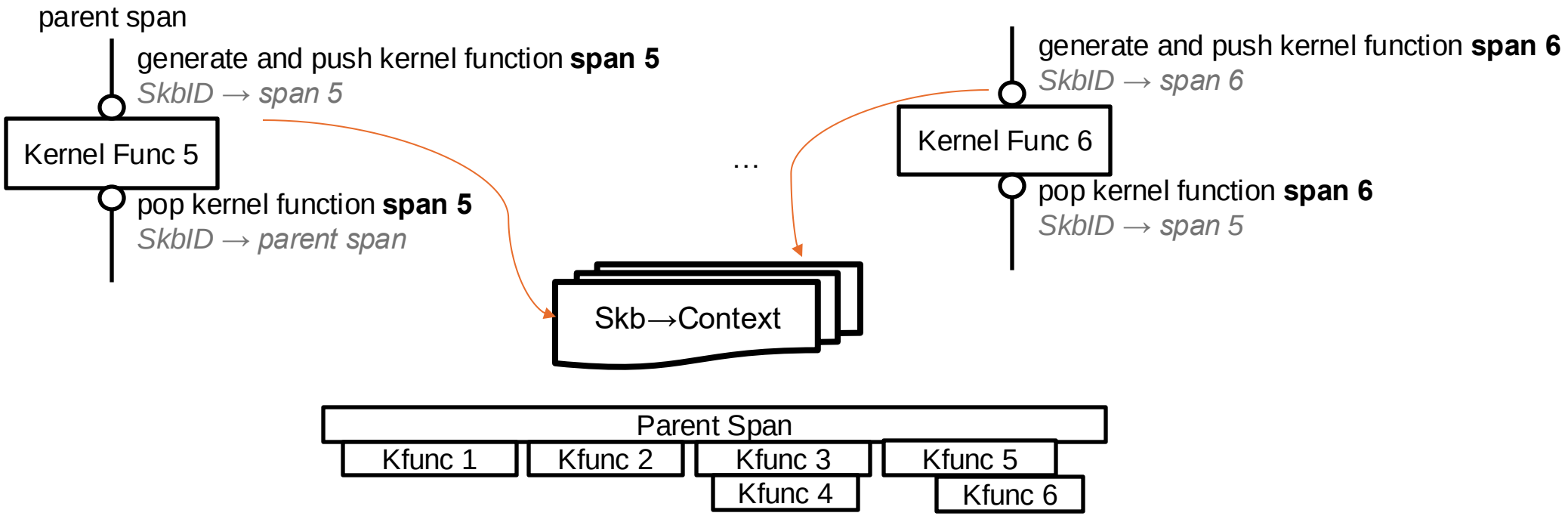
(1) sequential calls of kernel functions



(2) nested calls of kernel functions

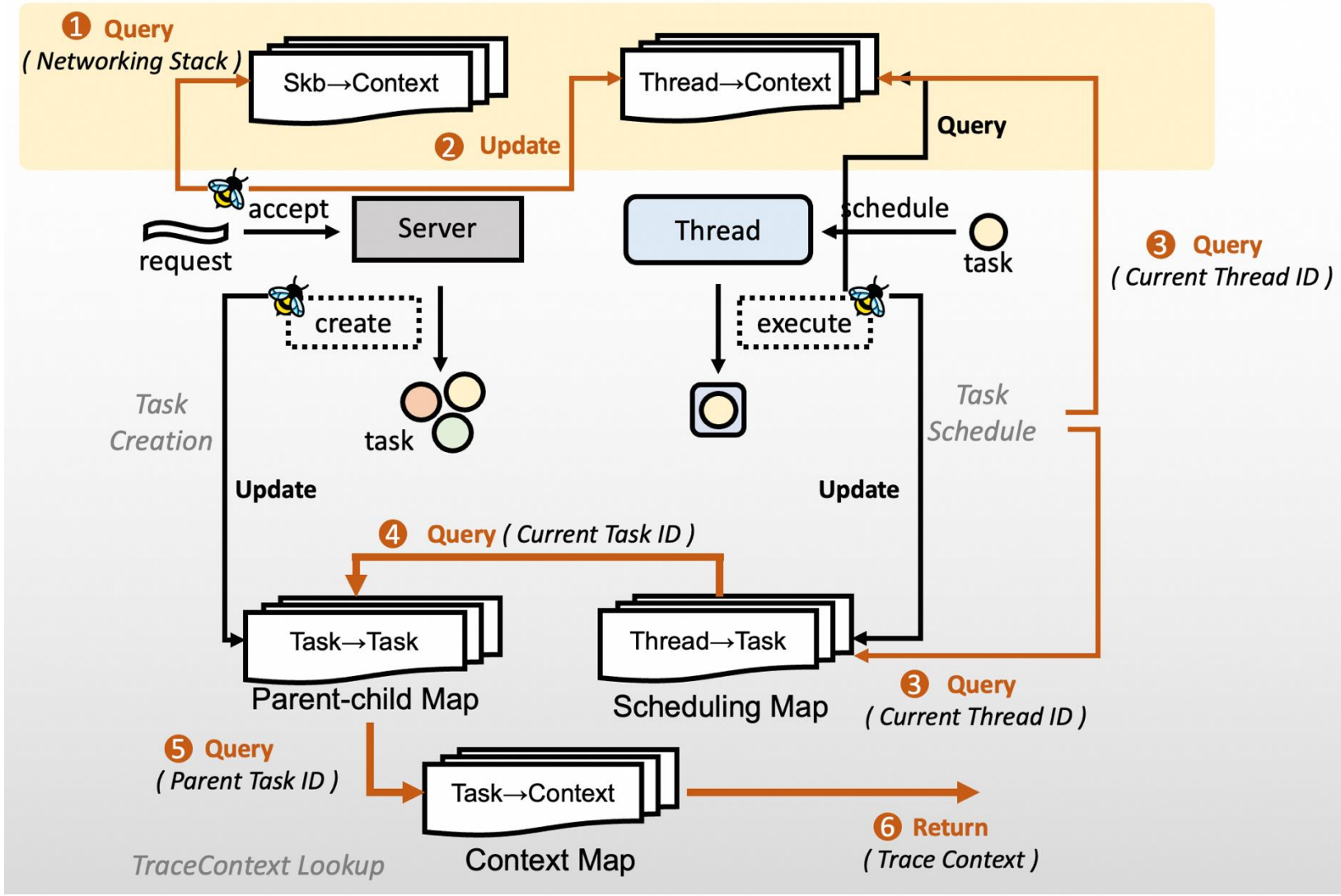
Context Propagation

Asynchronous Thread Model



(3) async calls of kernel functions in networking stack

Integrate kernel and request span



Example Demo

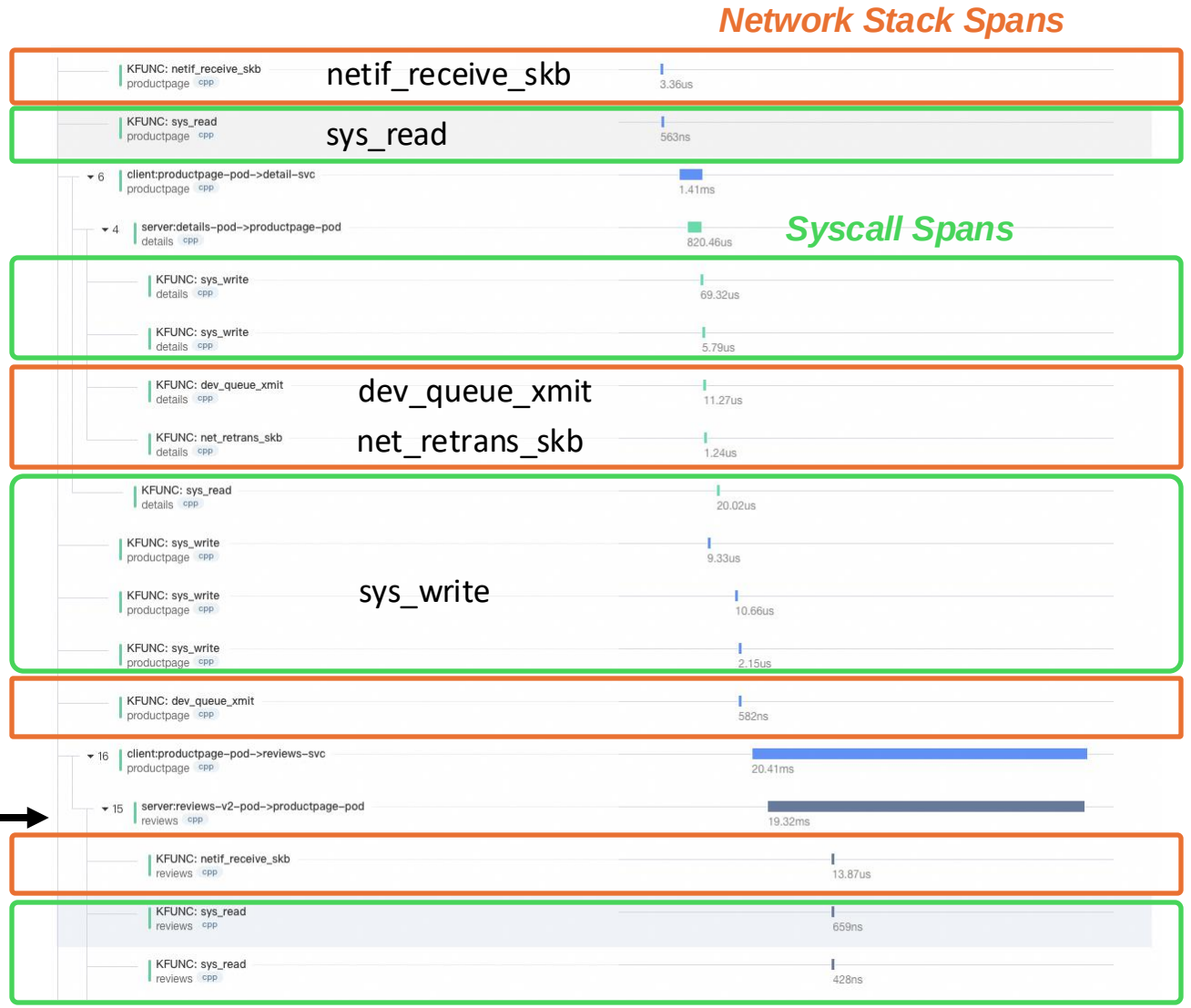
- Trace Visualization

Request-level insights

Network insights

Security insights

Before



After

Agenda

1. Observability in Kubernetes

2. End-to-End Request Tracing

3. Fine-grained Traces

4. Limitations & Future Works

Limitations

- Kernel Version >= 4.20
- Only supports W3C standard
- Only supports non-encrypted data
- Stream protocols currently are not supported

Future works

- Support context propagation across other communication protocols, like MySQL / Kafka
- Support NIC tracing to provide deeper network insights
- Enable continuous profiling to analyze the performance
- Integrate into Alibaba Cloud Application Monitoring eBPF Edition
- Contribute the code to community


KubeCon
CloudNativeCon
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
AI_dev
Open Source GenAI & ML Summit
China 2024
Thank you !