Skua: Extending Distributed Tracing Vertically into the Linux Kernel

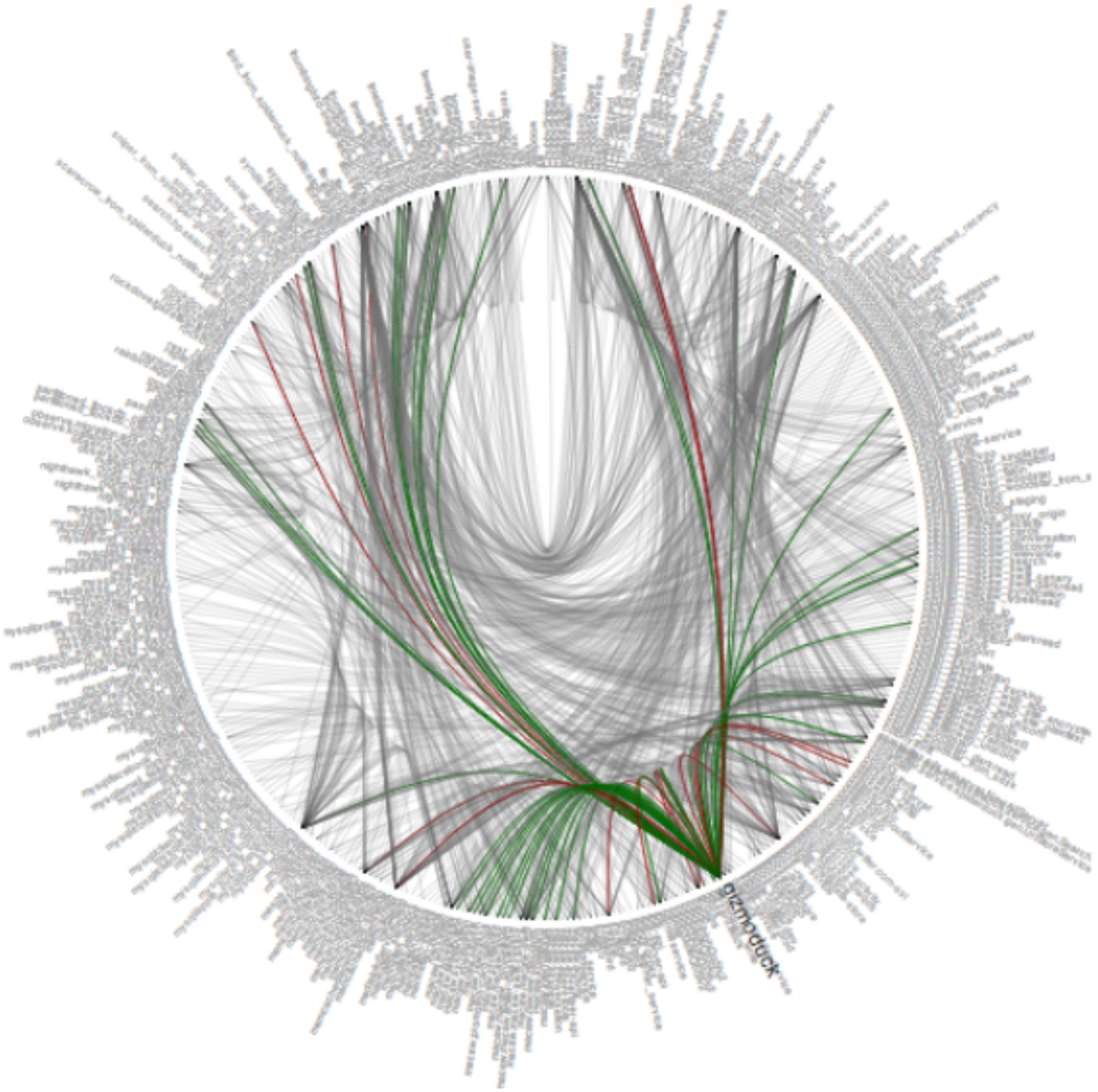
Harshal Sheth and Andrew Sun

Distributed Systems

- Complex applications are no longer monolithic
    - Modular/agile development
    - Continuous deployment
    - Independent scaling

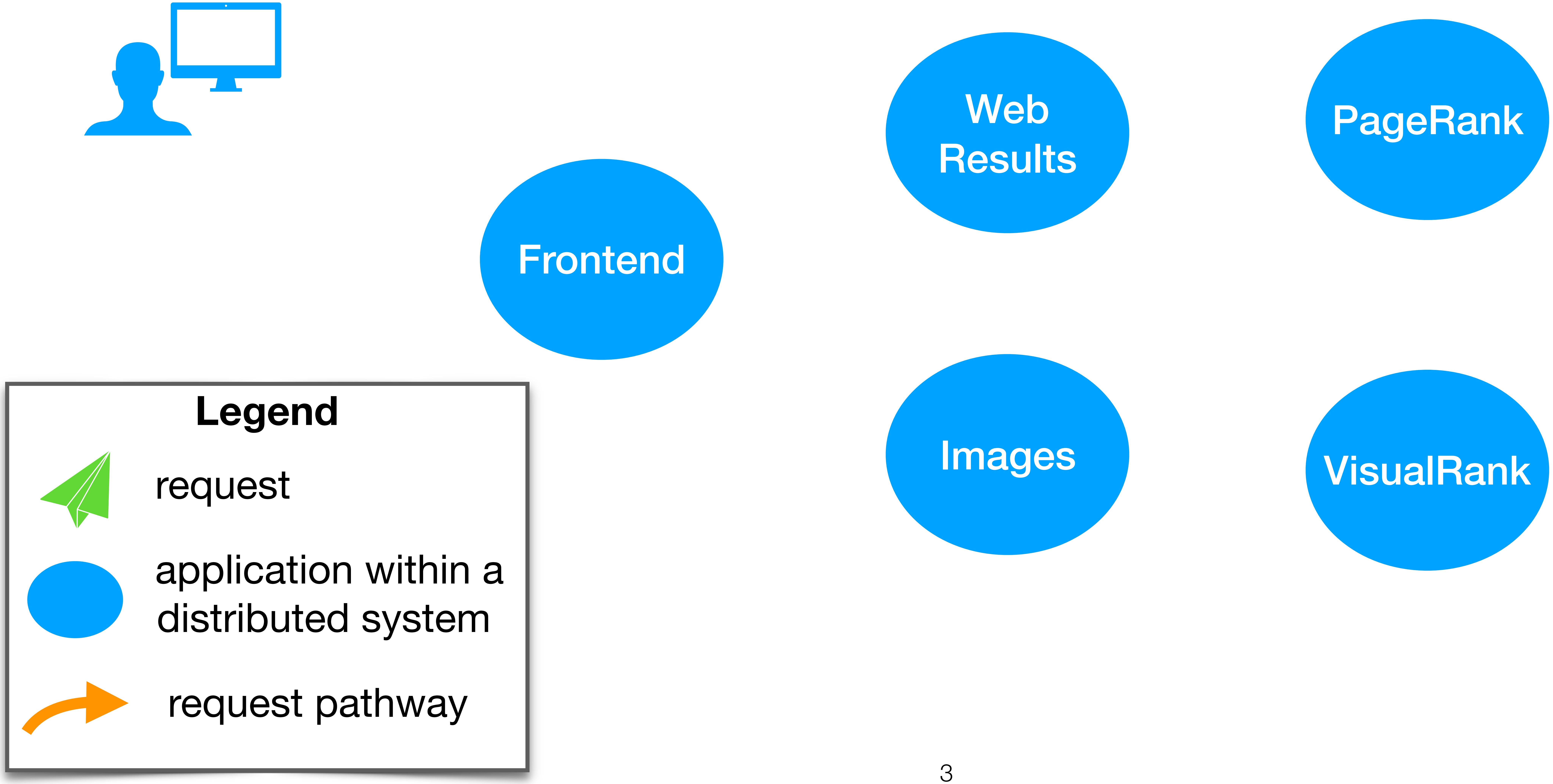
Distributed Systems

- Complex applications are no longer monolithic
  - Modular/agile development
  - Continuous deployment
  - Independent scaling
- Increasingly seen in large companies
- Hard to debug

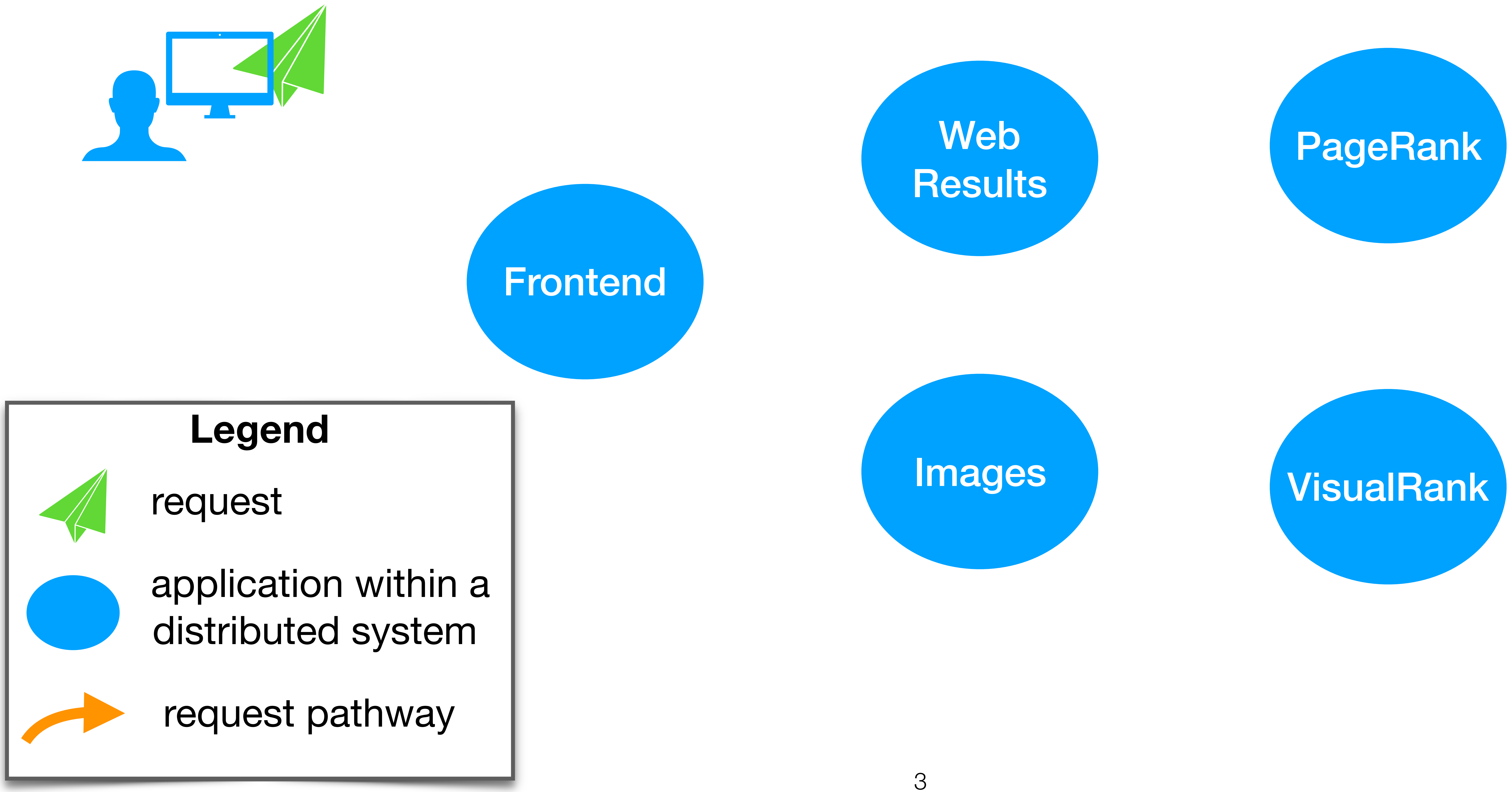


Twitter, 2013

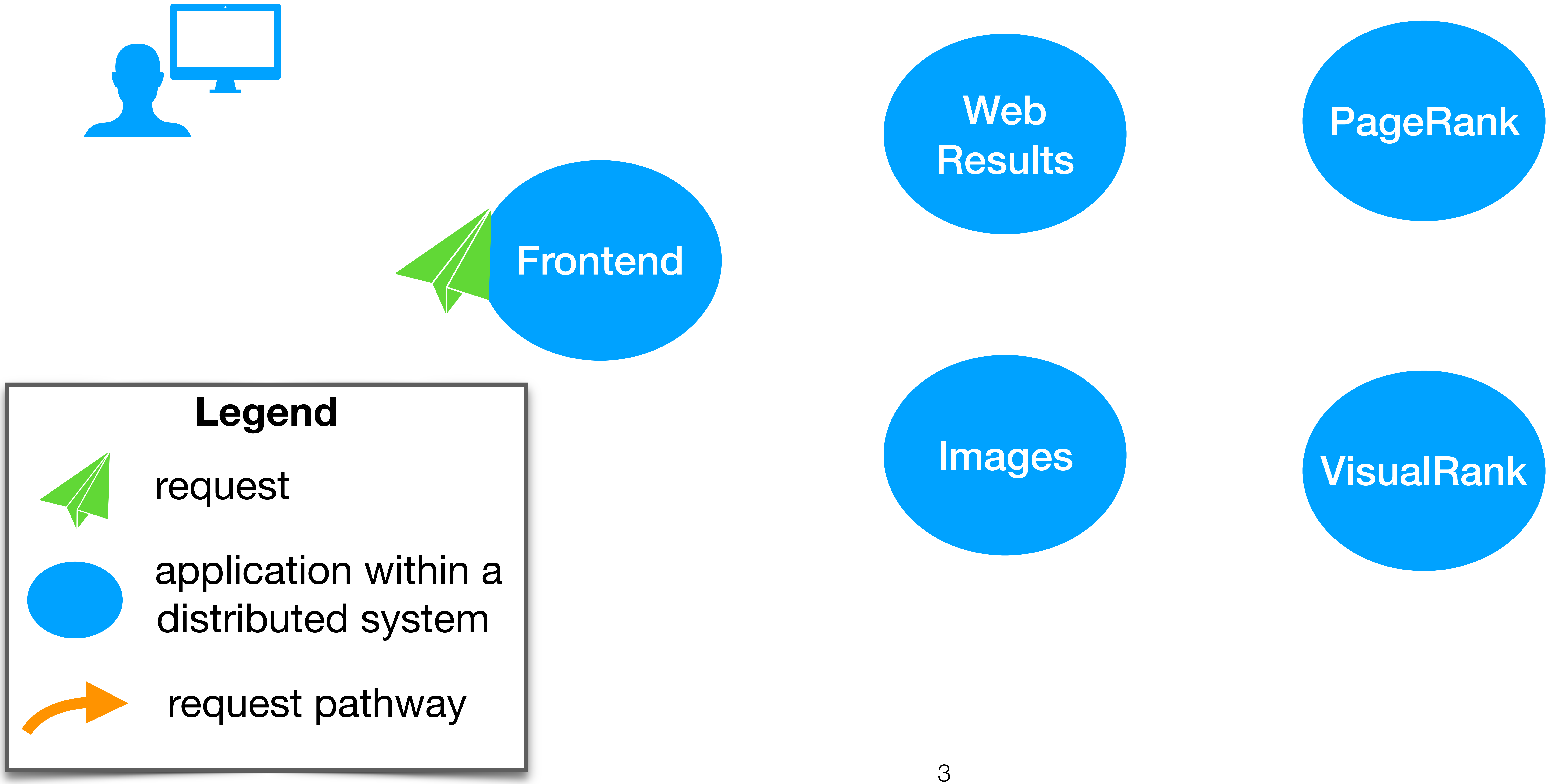
Example Distributed System



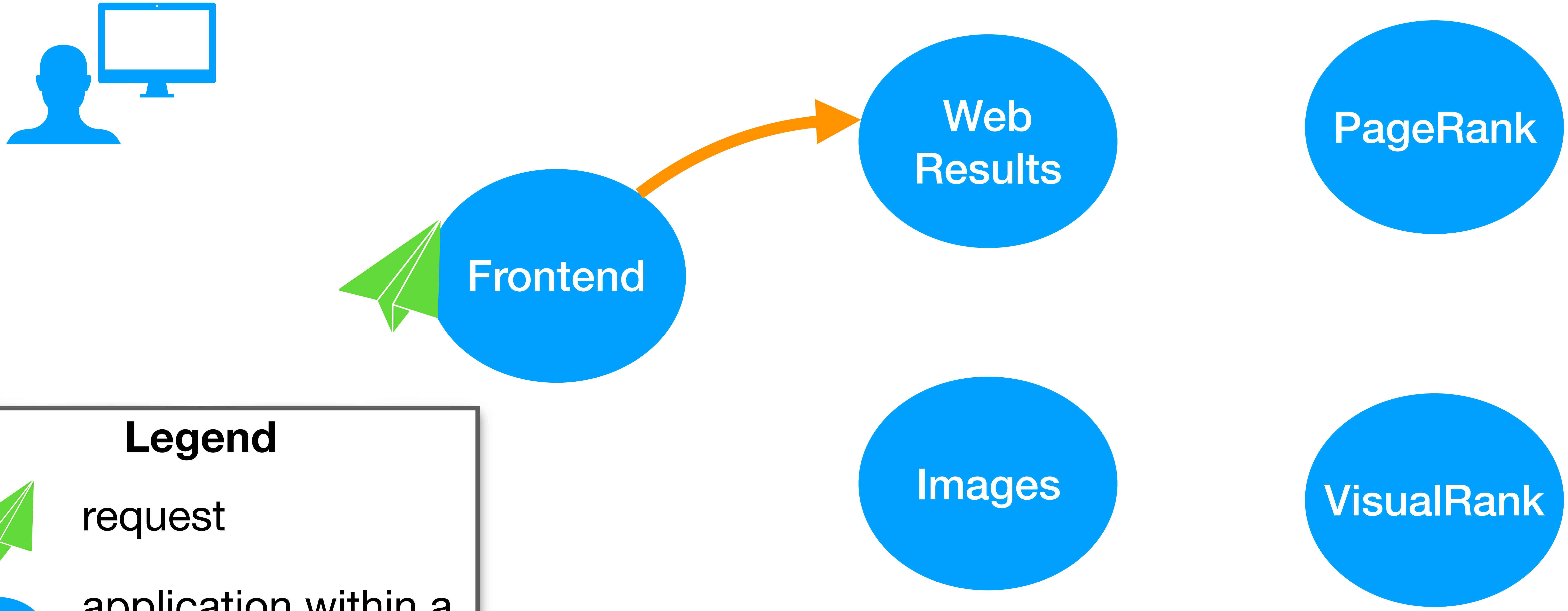
Example Distributed System



Example Distributed System



Example Distributed System



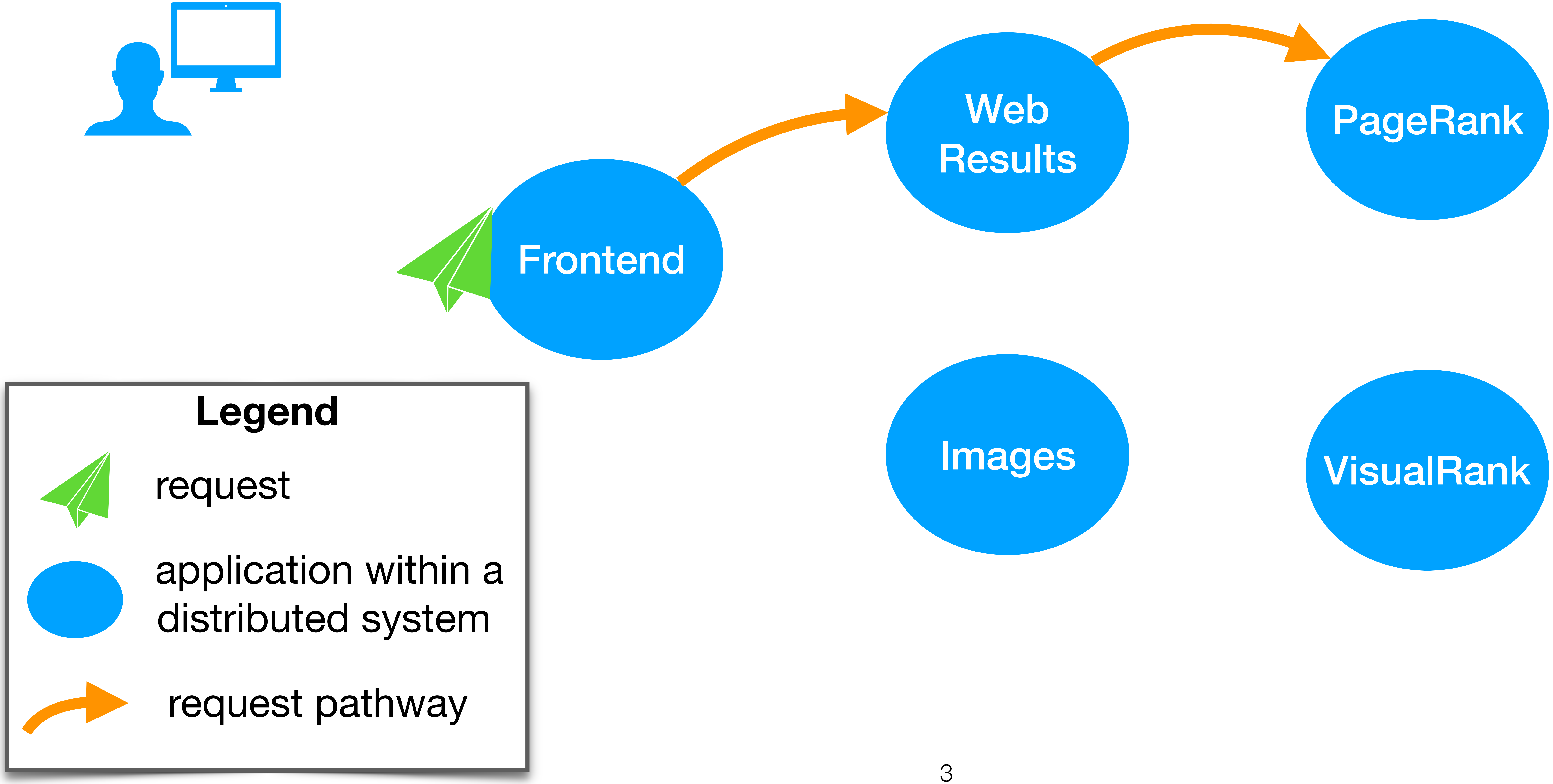
Legend

request

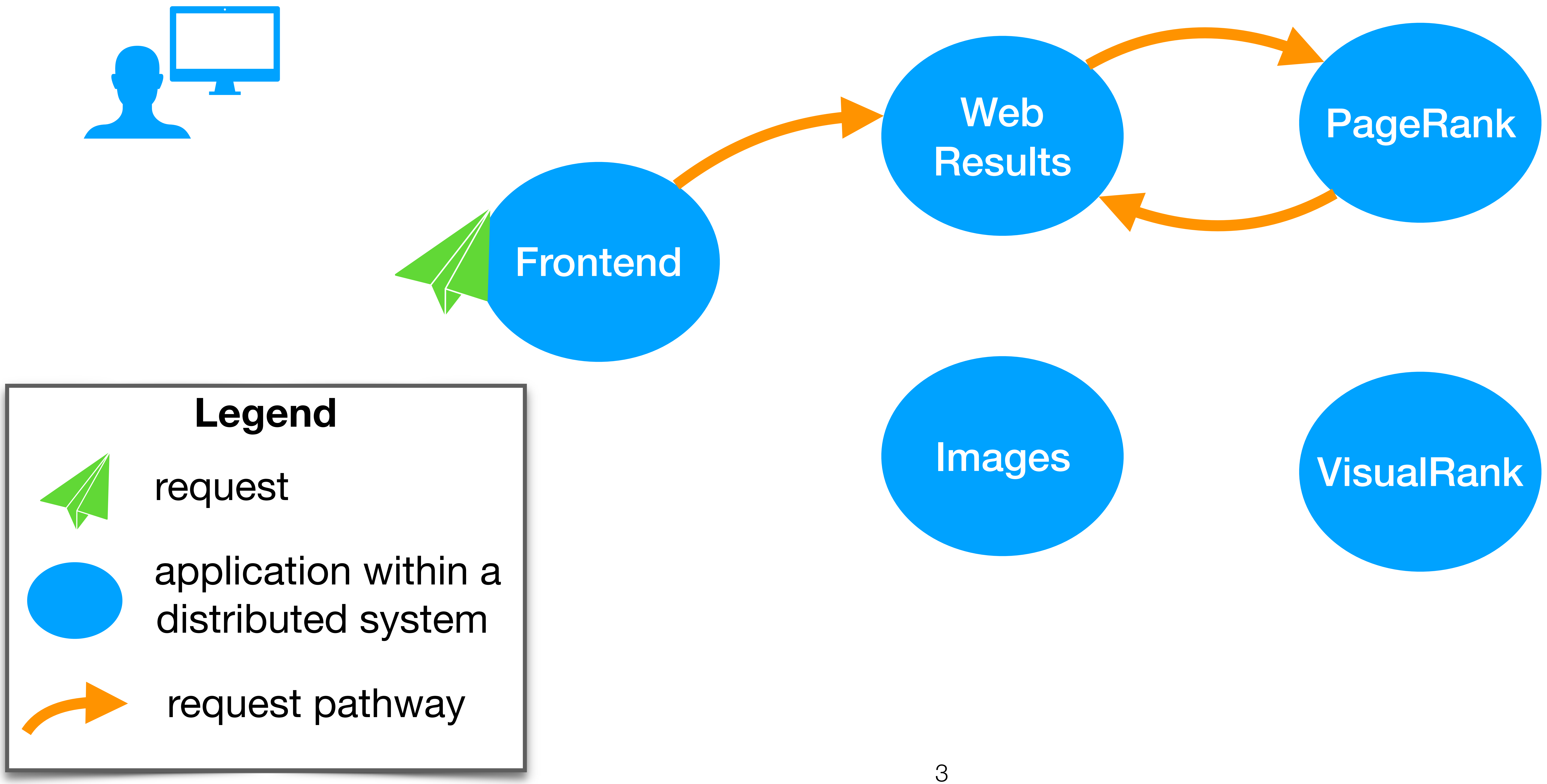
application within a distributed system

request pathway

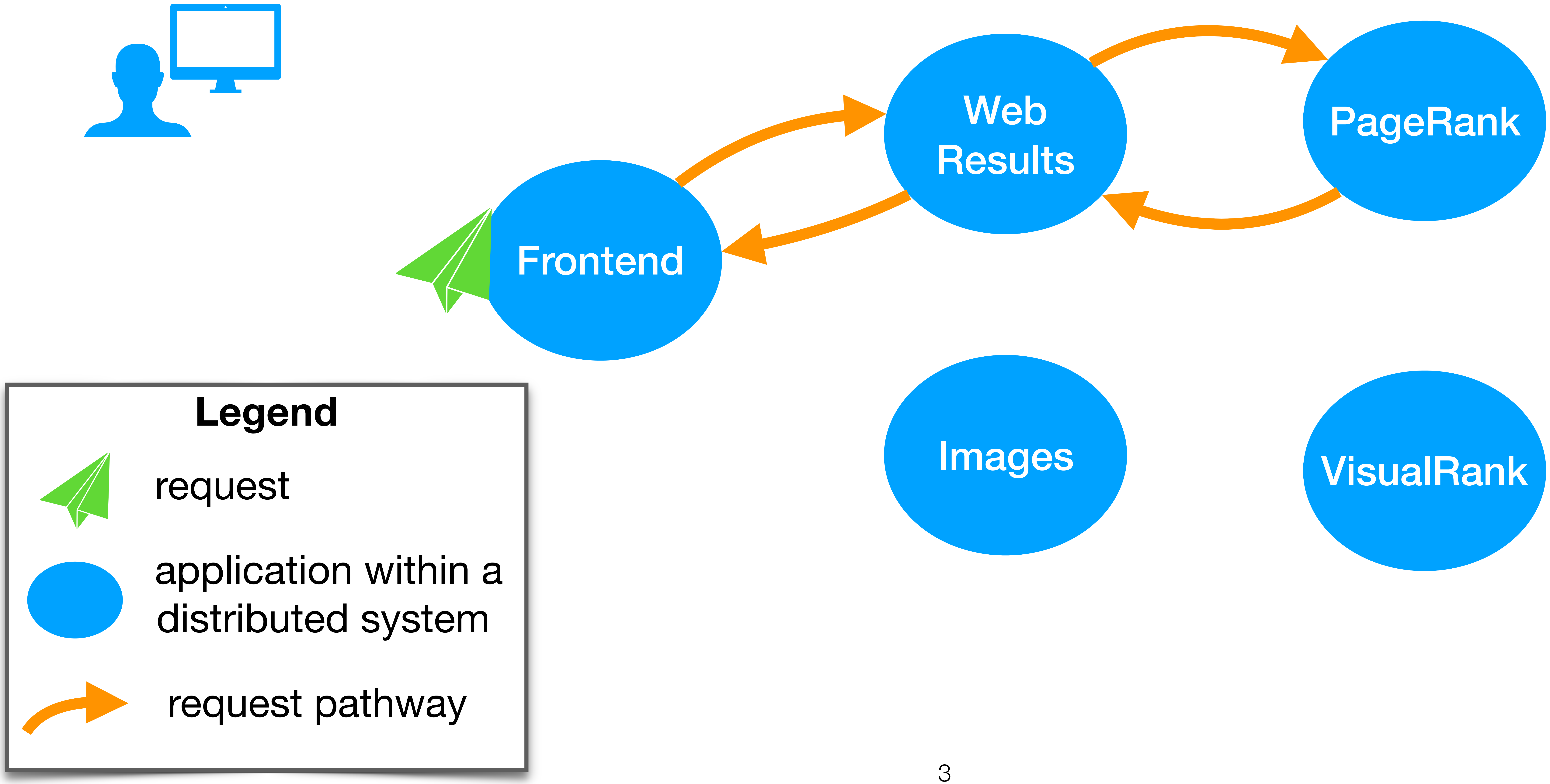
Example Distributed System



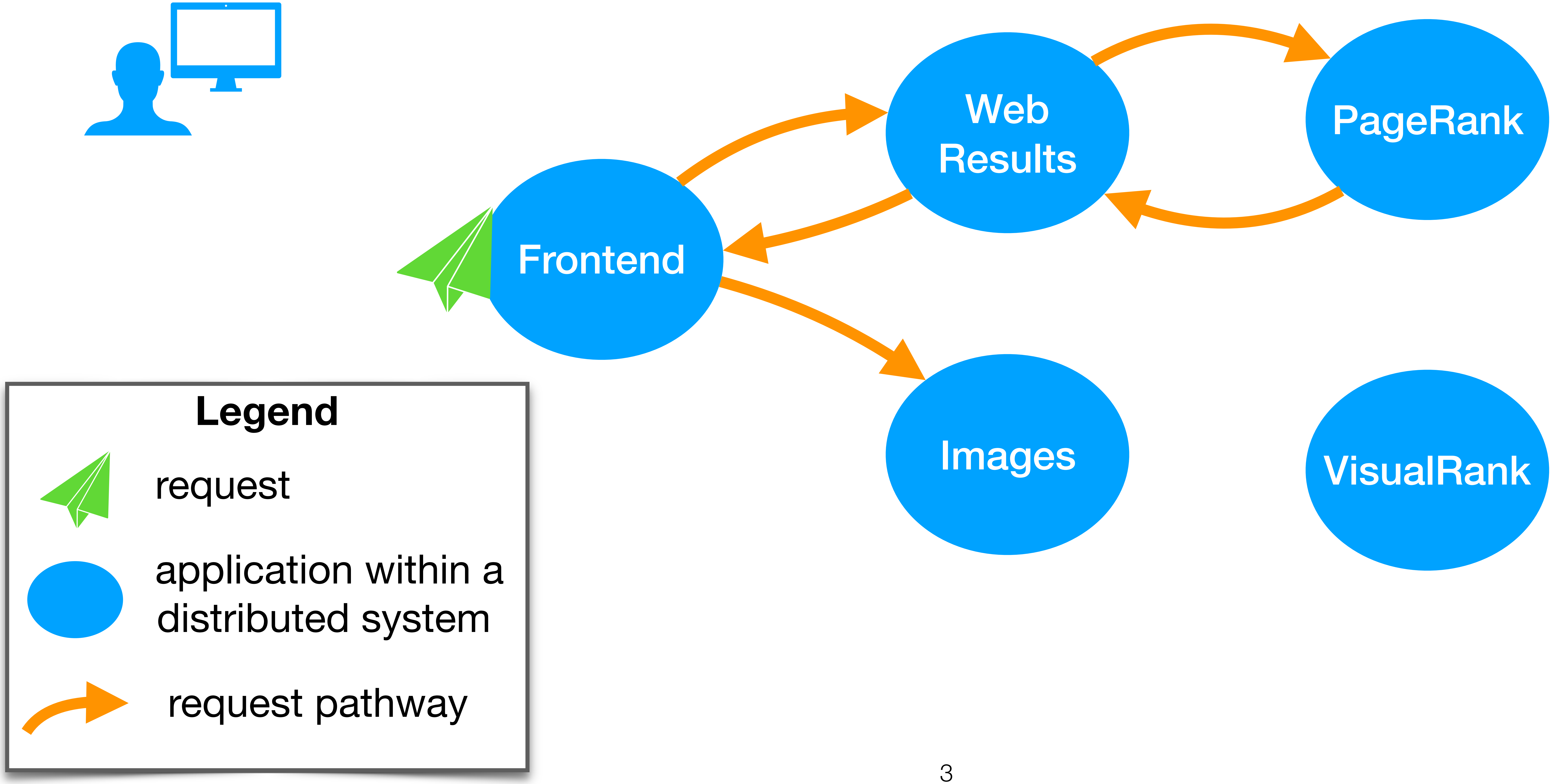
Example Distributed System



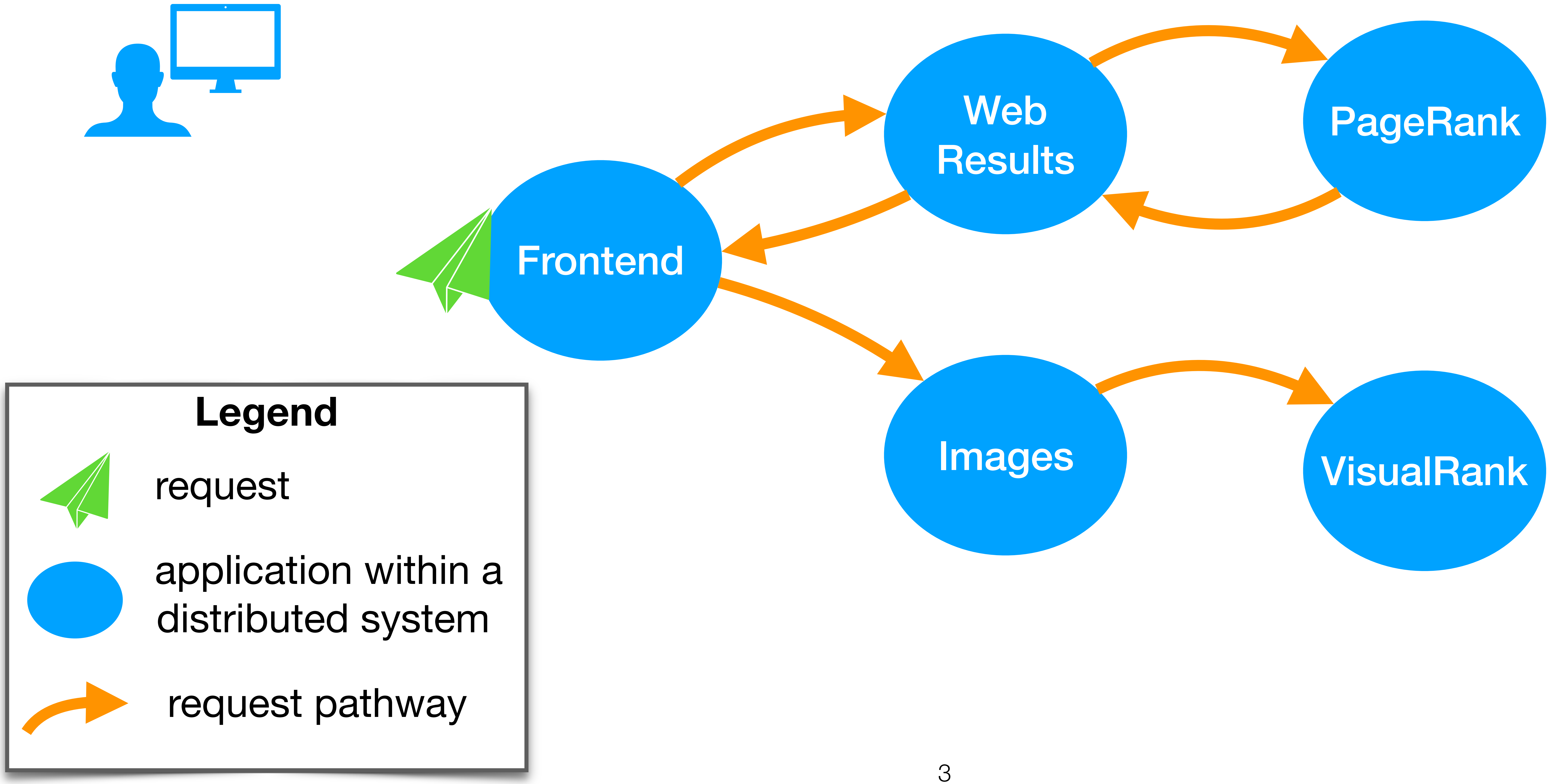
Example Distributed System



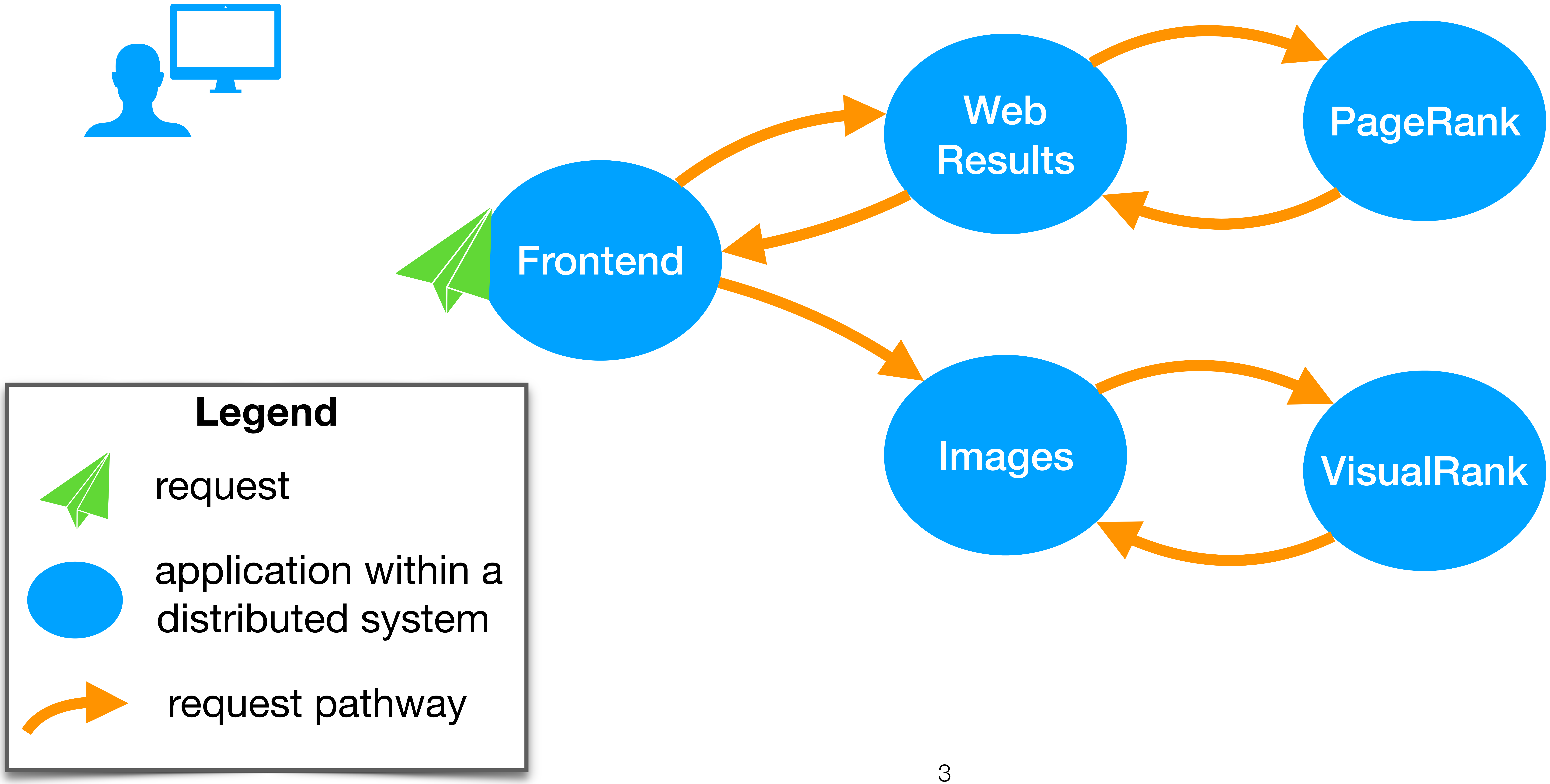
Example Distributed System



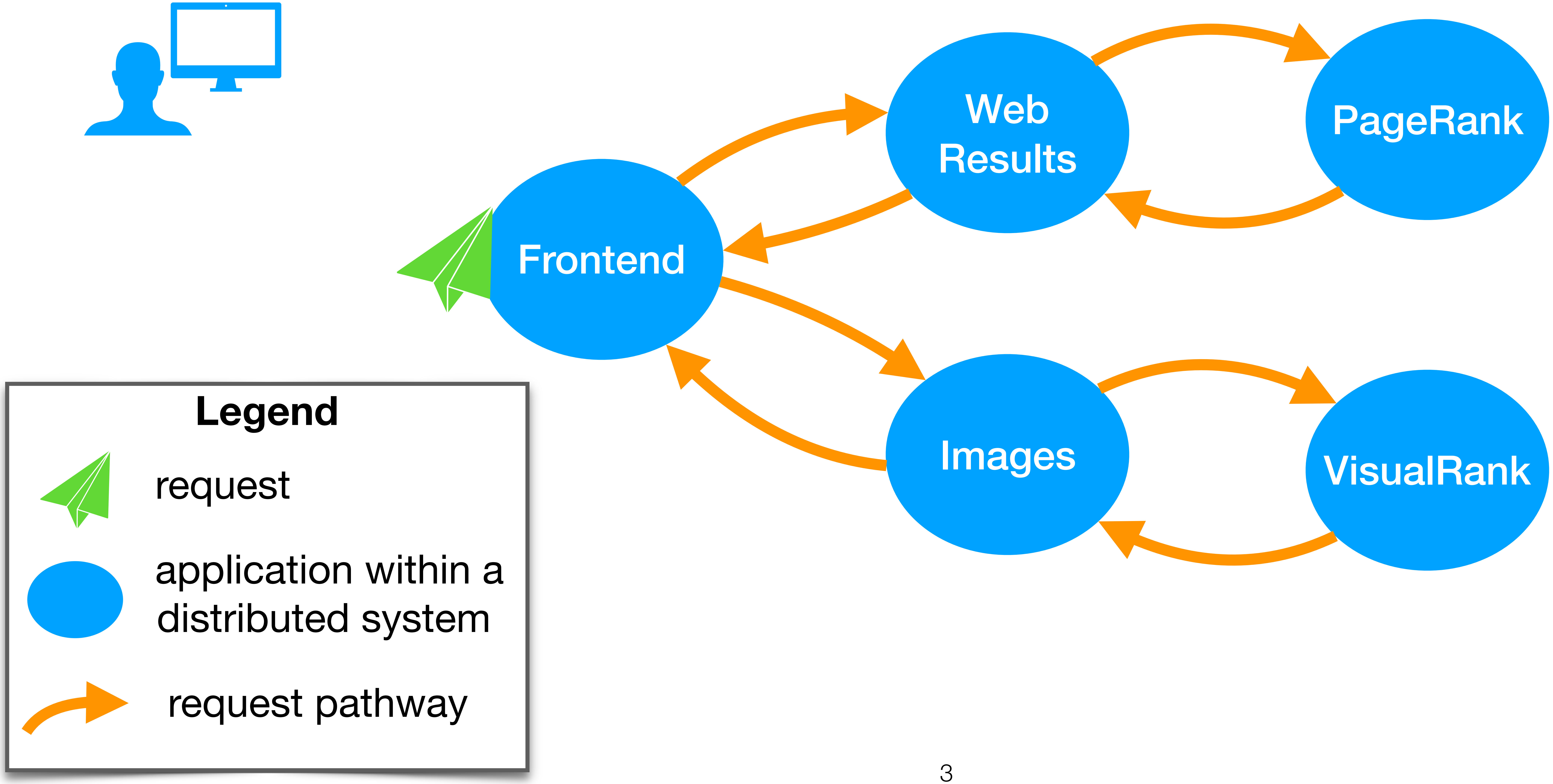
Example Distributed System



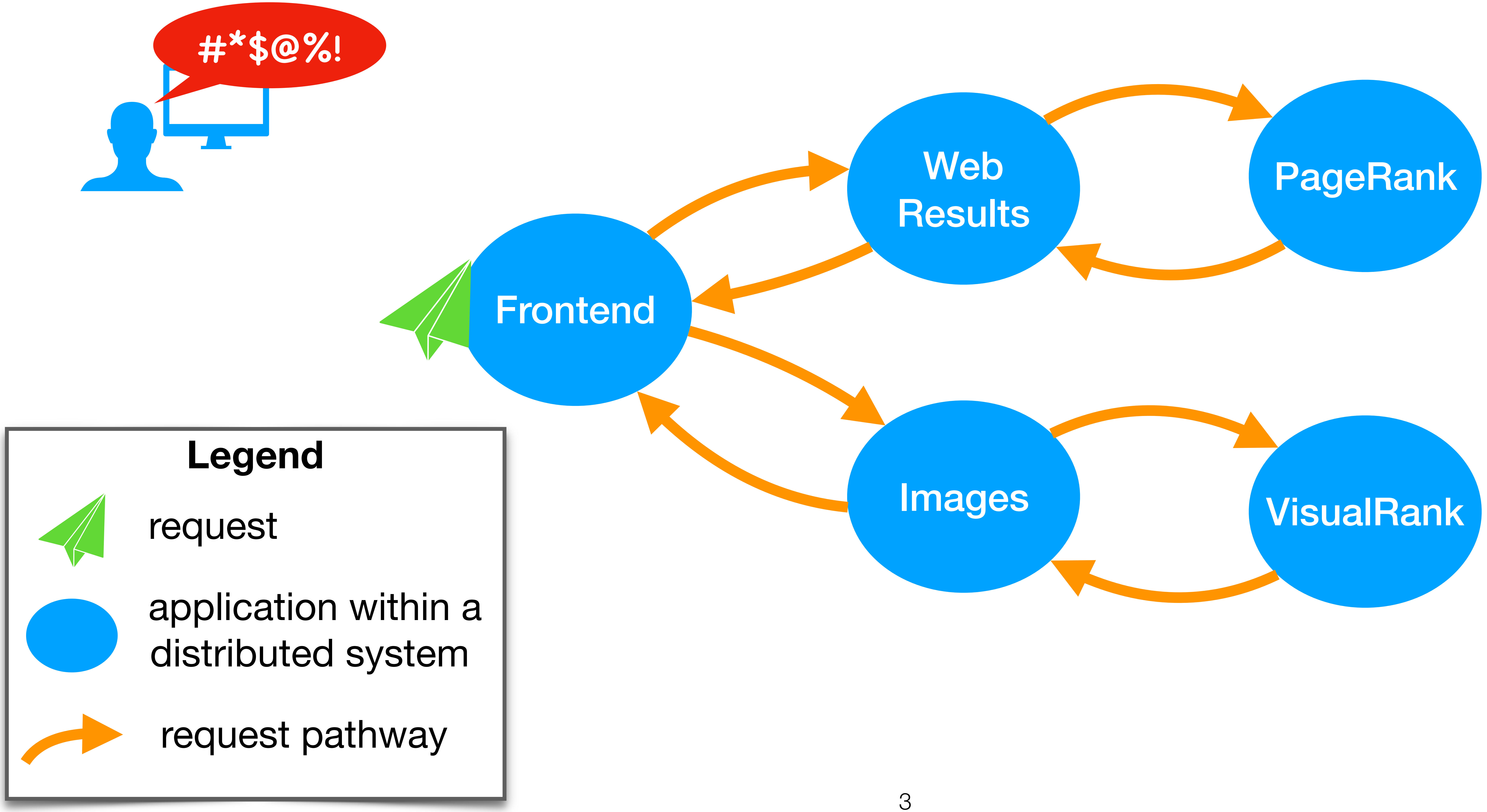
Example Distributed System



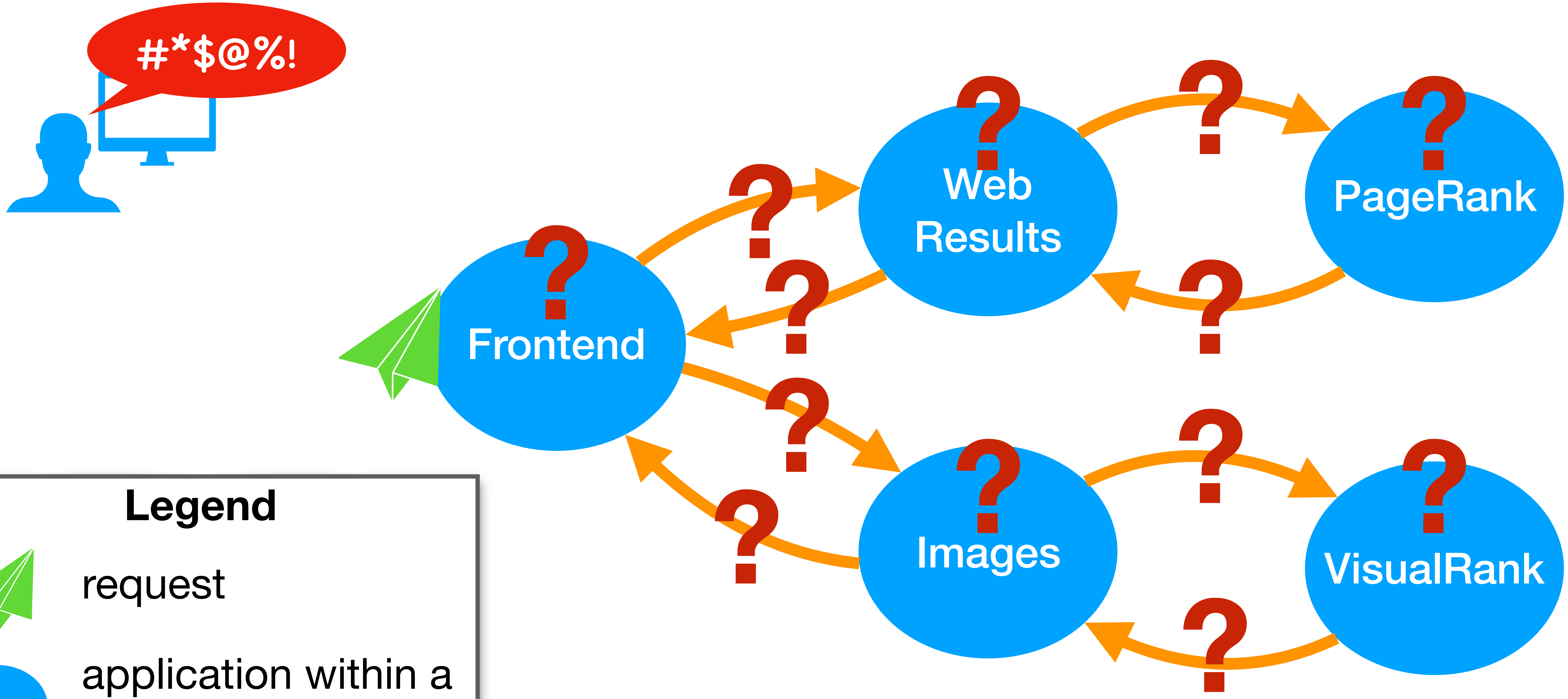
Example Distributed System

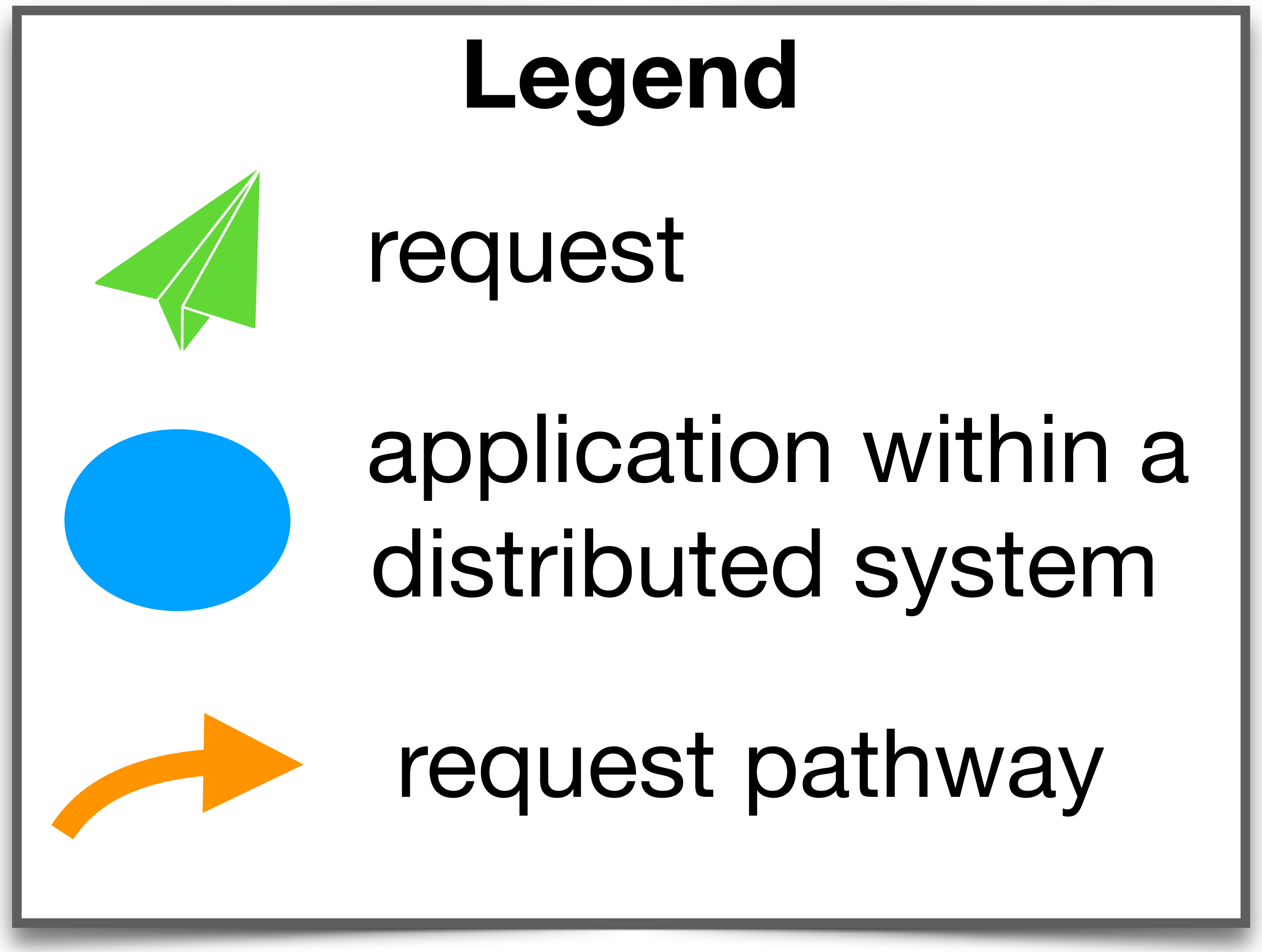


Example Distributed System



Example Distributed System





Distributed Tracing

- Monitoring and troubleshooting distributed systems

Distributed Tracing

- Monitoring and troubleshooting distributed systems
  - Discovering latency issues

Distributed Tracing

- Monitoring and troubleshooting distributed systems
    - Discovering latency issues
    - Graphing service dependencies

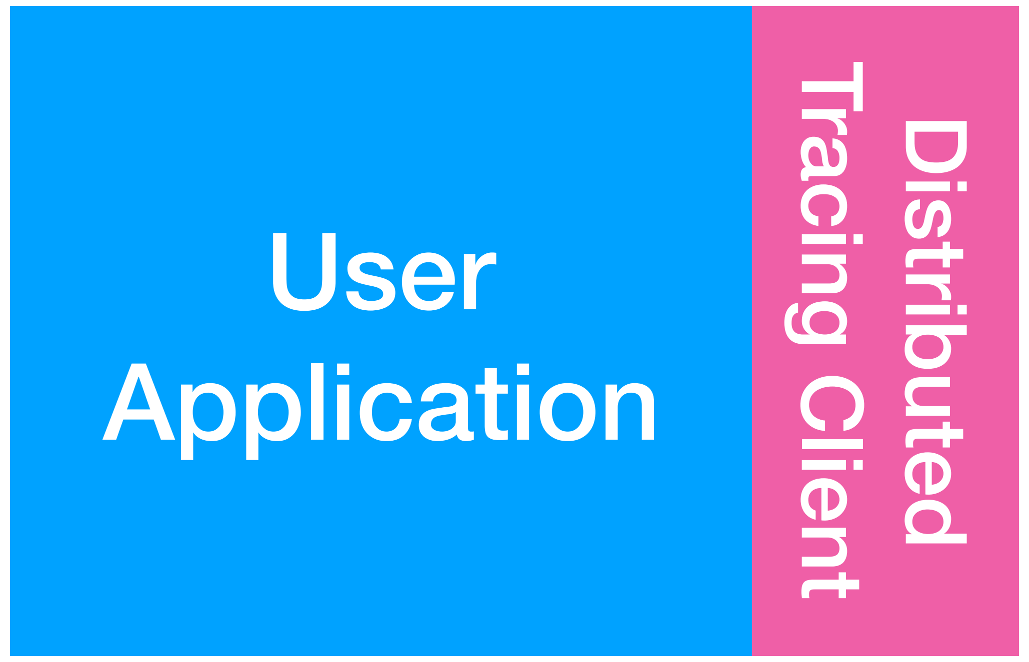
Distributed Tracing

- Monitoring and troubleshooting distributed systems
  - Discovering latency issues
  - Graphing service dependencies
  - Root-cause analysis of backend issues

Distributed Tracing

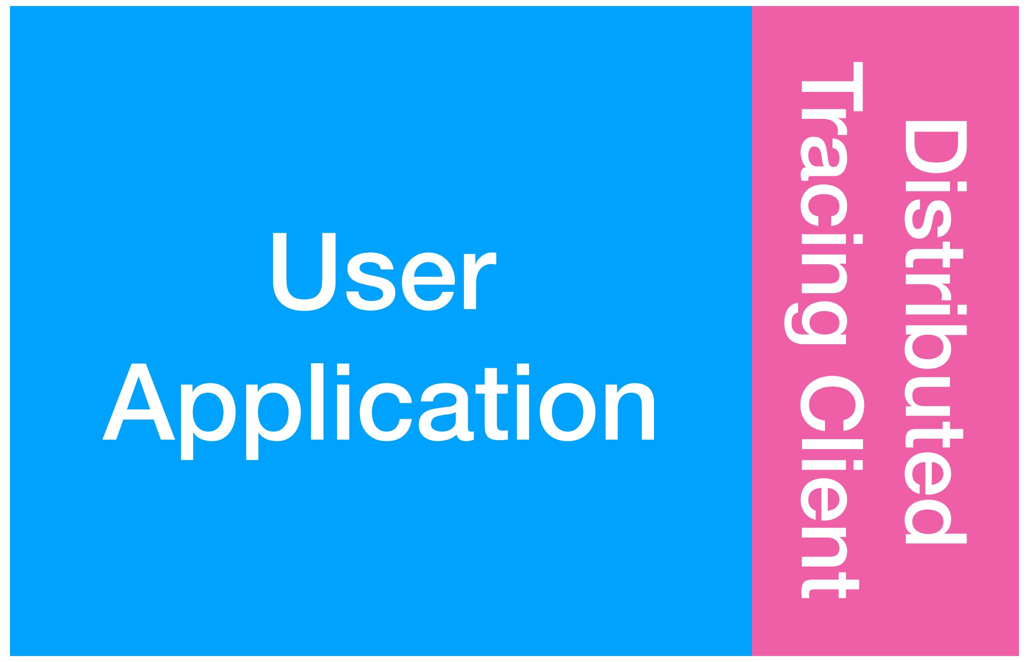
- Monitoring and troubleshooting distributed systems
  - Discovering latency issues
  - Graphing service dependencies
  - Root-cause analysis of backend issues
  - Tracing a specific request through the entire system

What does distributed tracing miss?



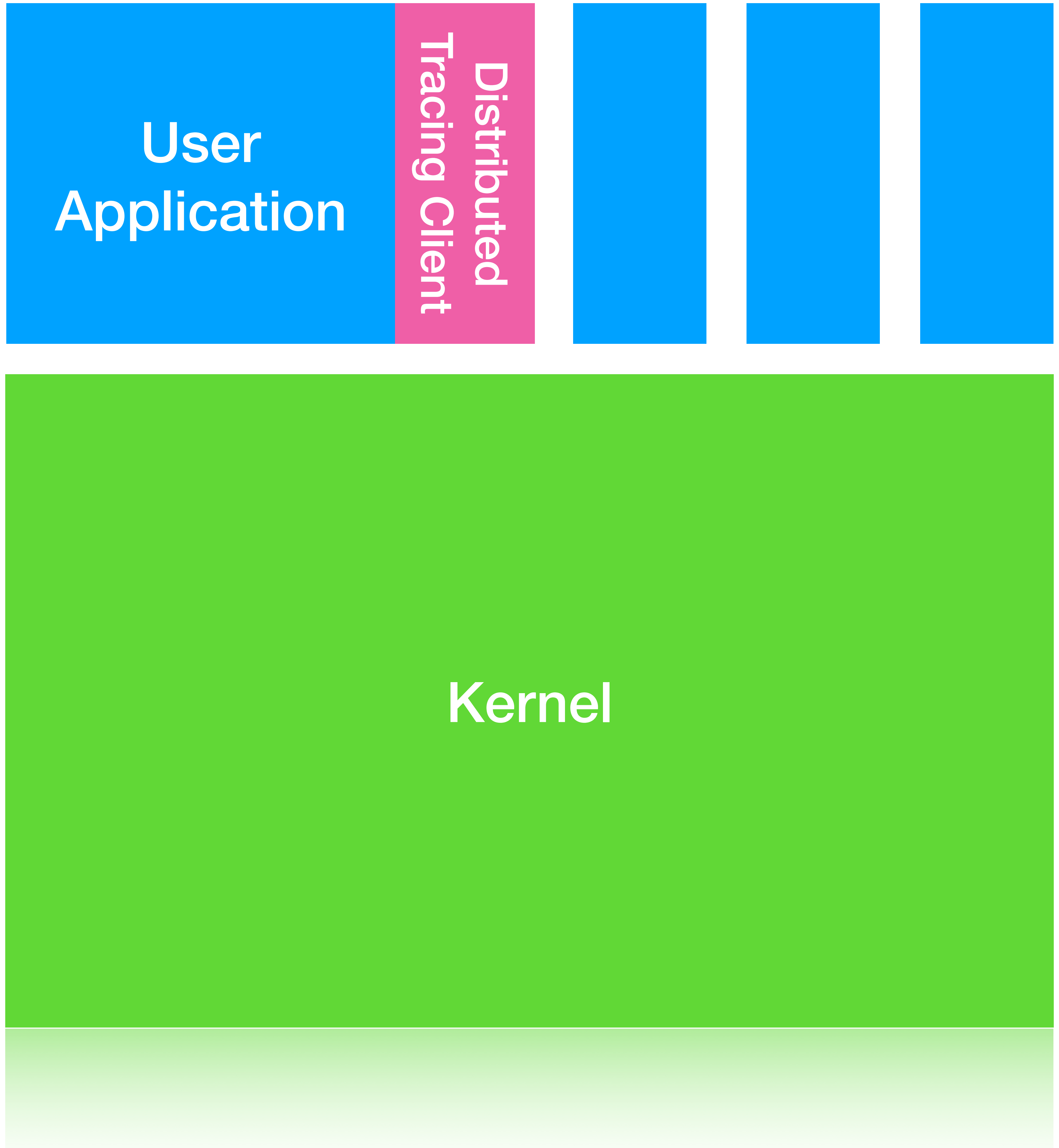
What does distributed tracing miss?

- There's more to performance than meets the eye of existing distributed tracing tools



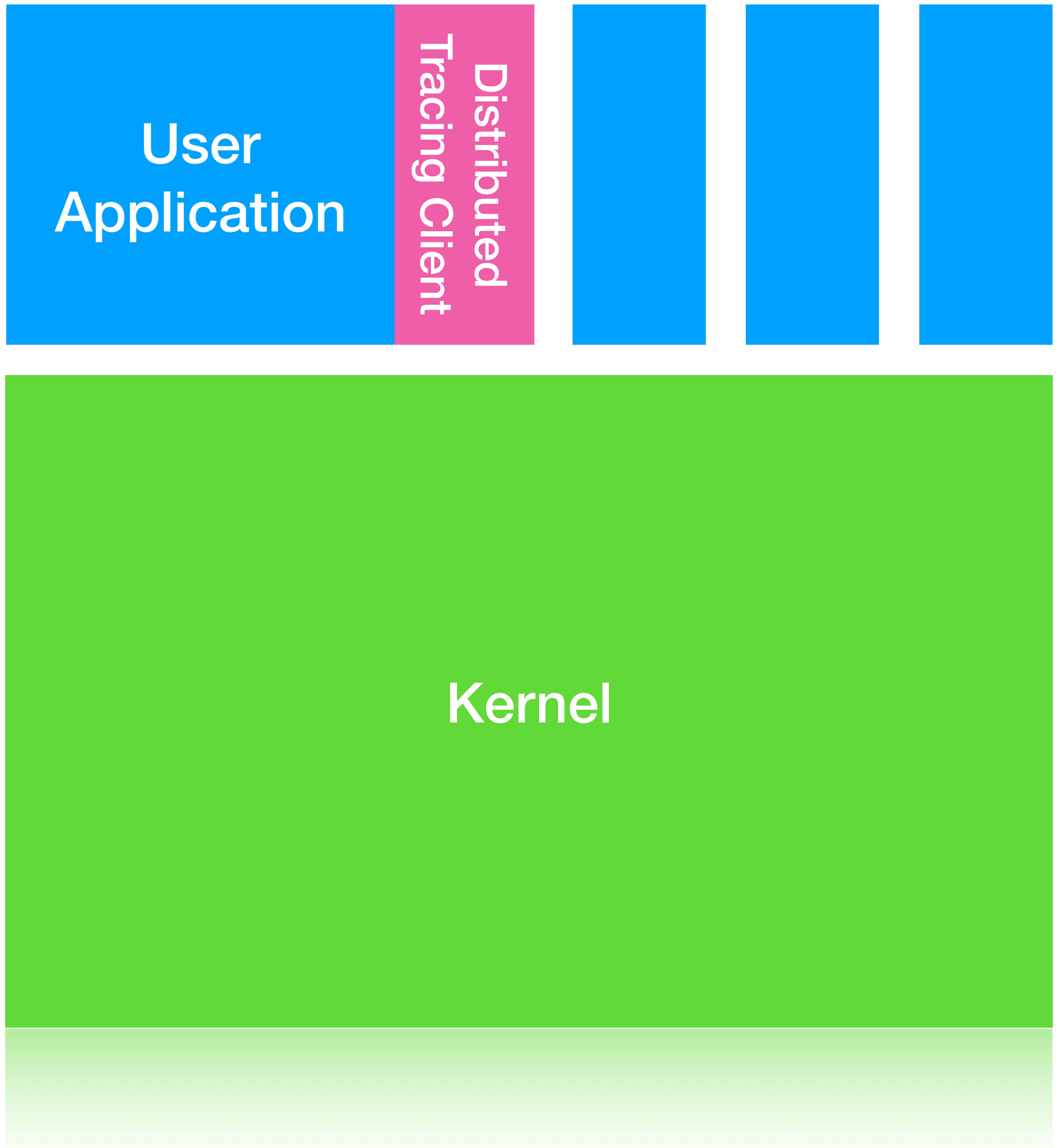
What does distributed tracing miss?

- There's more to performance than meets the eye of existing distributed tracing tools
  - Contention between applications
  - Kernel bugs
  - Security patches (e.g. Meltdown/Spectre)



What does distributed tracing miss?

- There's more to performance than meets the eye of existing distributed tracing tools
  - Contention between applications
  - Kernel bugs
  - Security patches (e.g. Meltdown/Spectre)
- Can we gain visibility regarding these issues via the kernel?



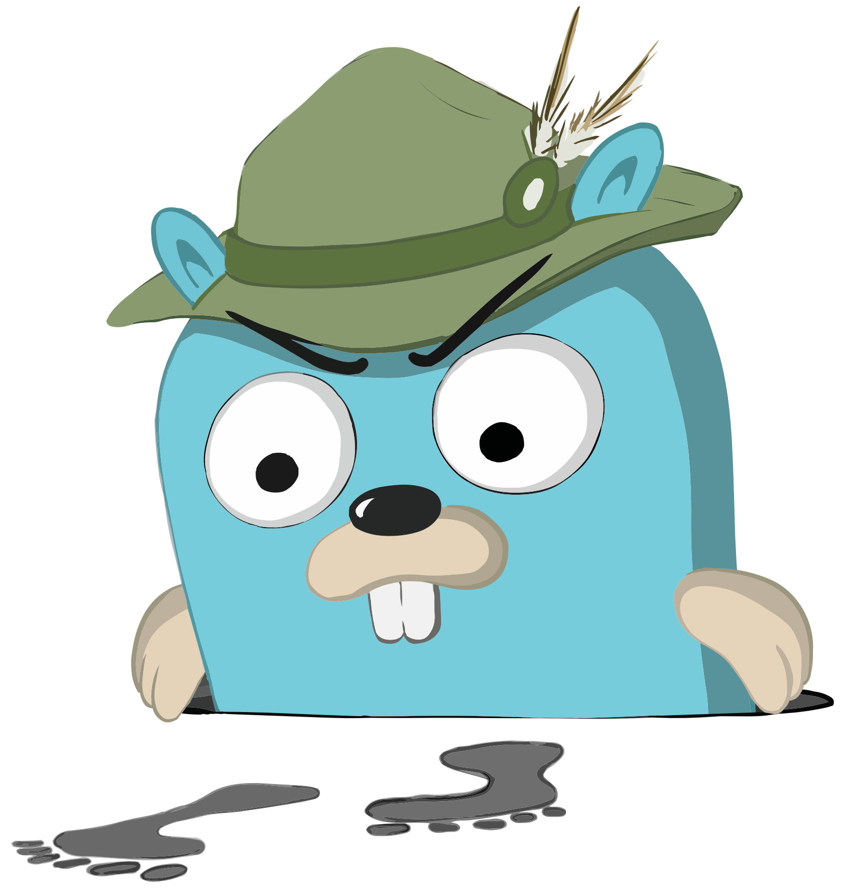
Our goal: extend distributed tracing into the kernel

Our Approach

Jaeger
distributed tracing
framework from Uber

Our Approach

Jaeger
distributed tracing
framework from Uber



+

LTTng
Linux kernel
trace toolkit

Our Approach

Jaeger
distributed tracing
framework from Uber

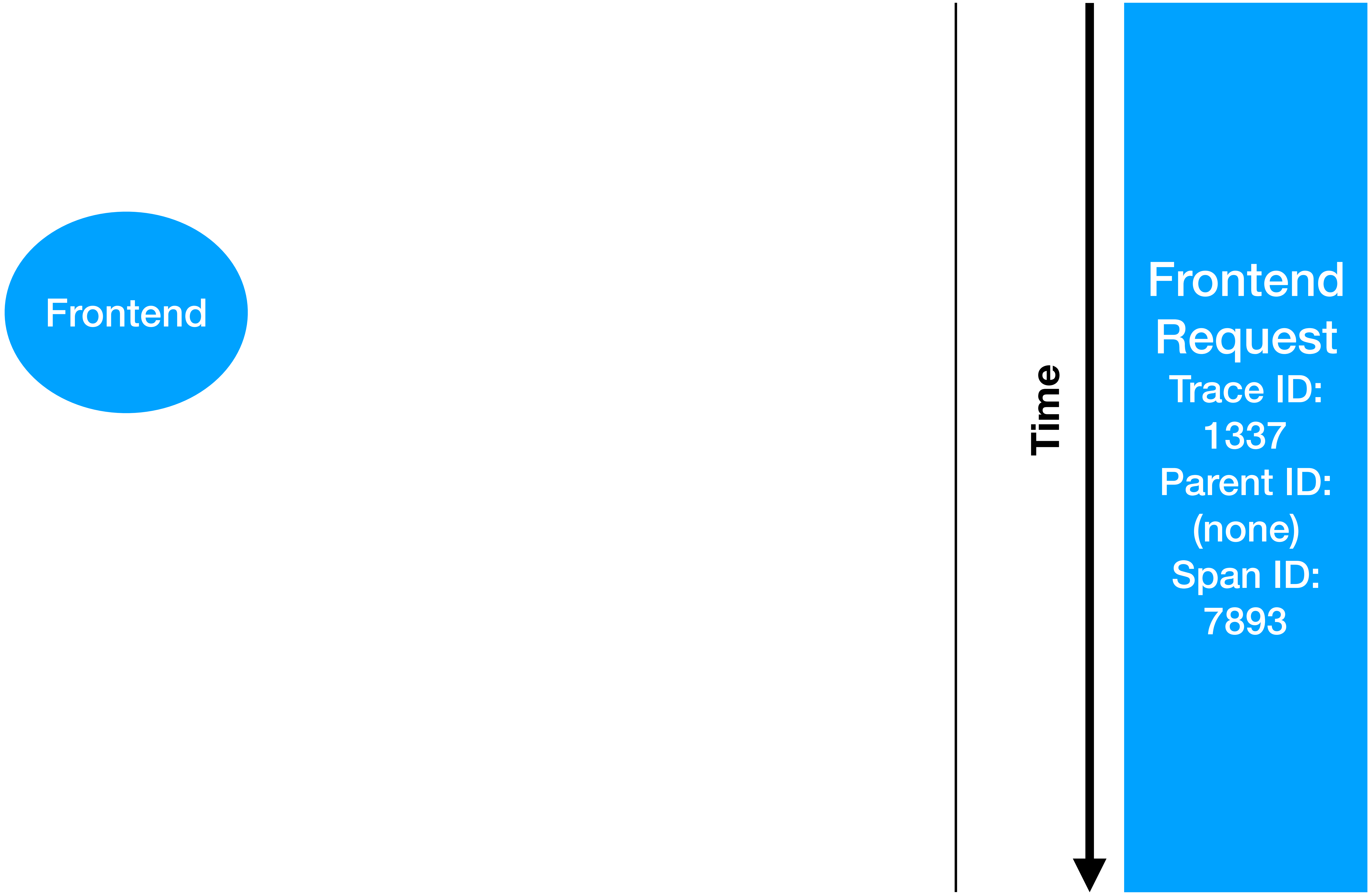
+

LTTng
Linux kernel
trace toolkit

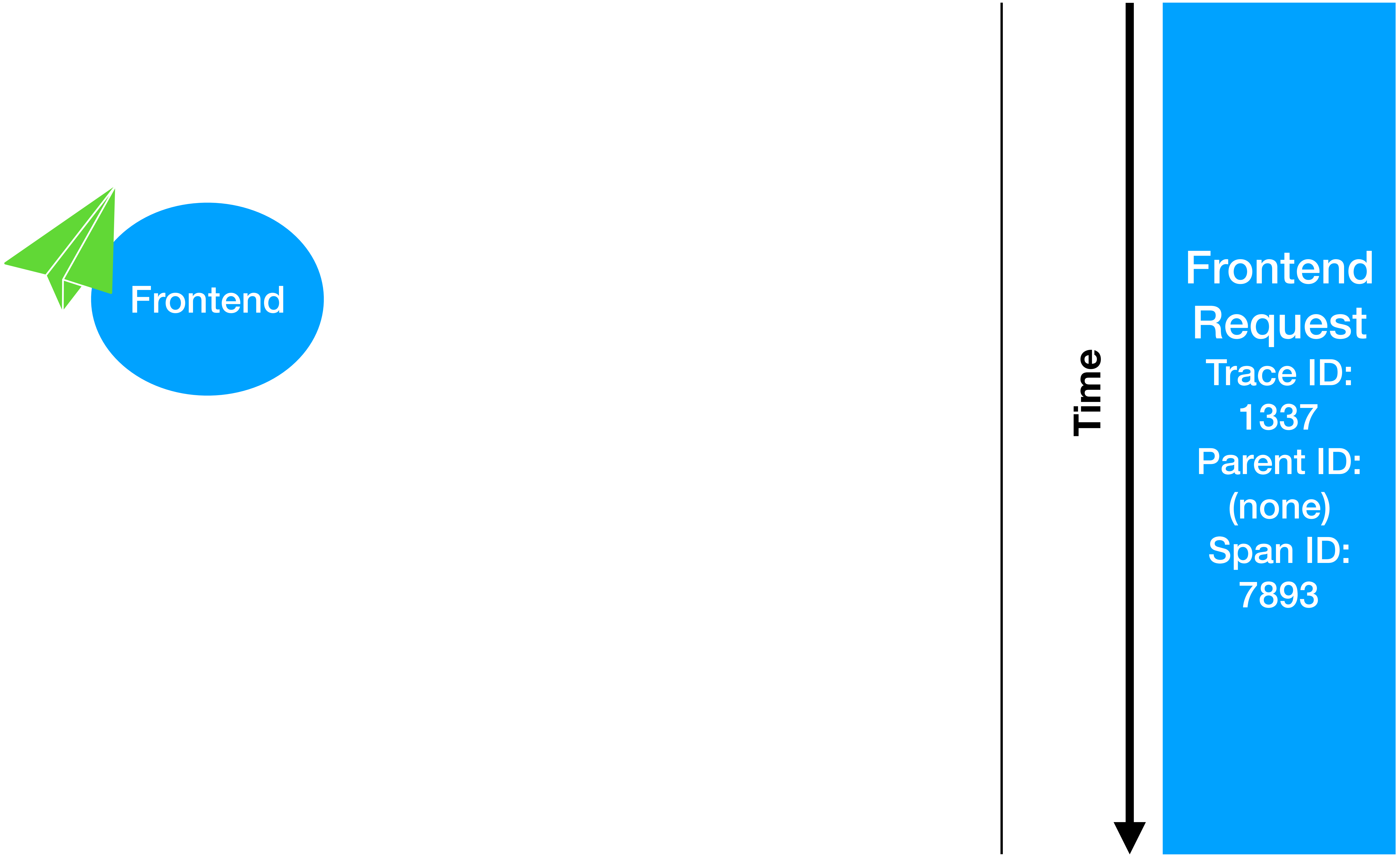
=

Skua

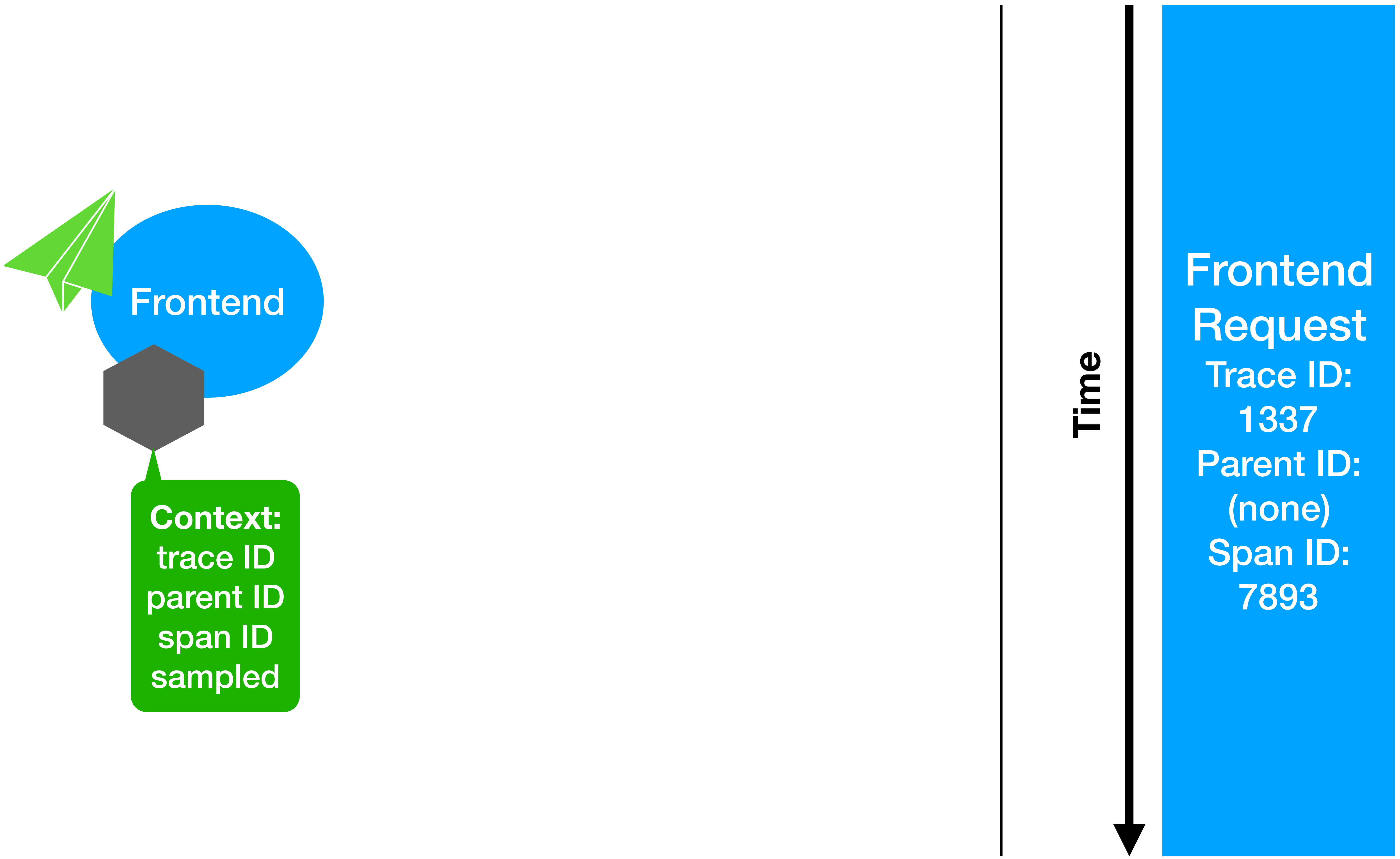
Traces and Spans



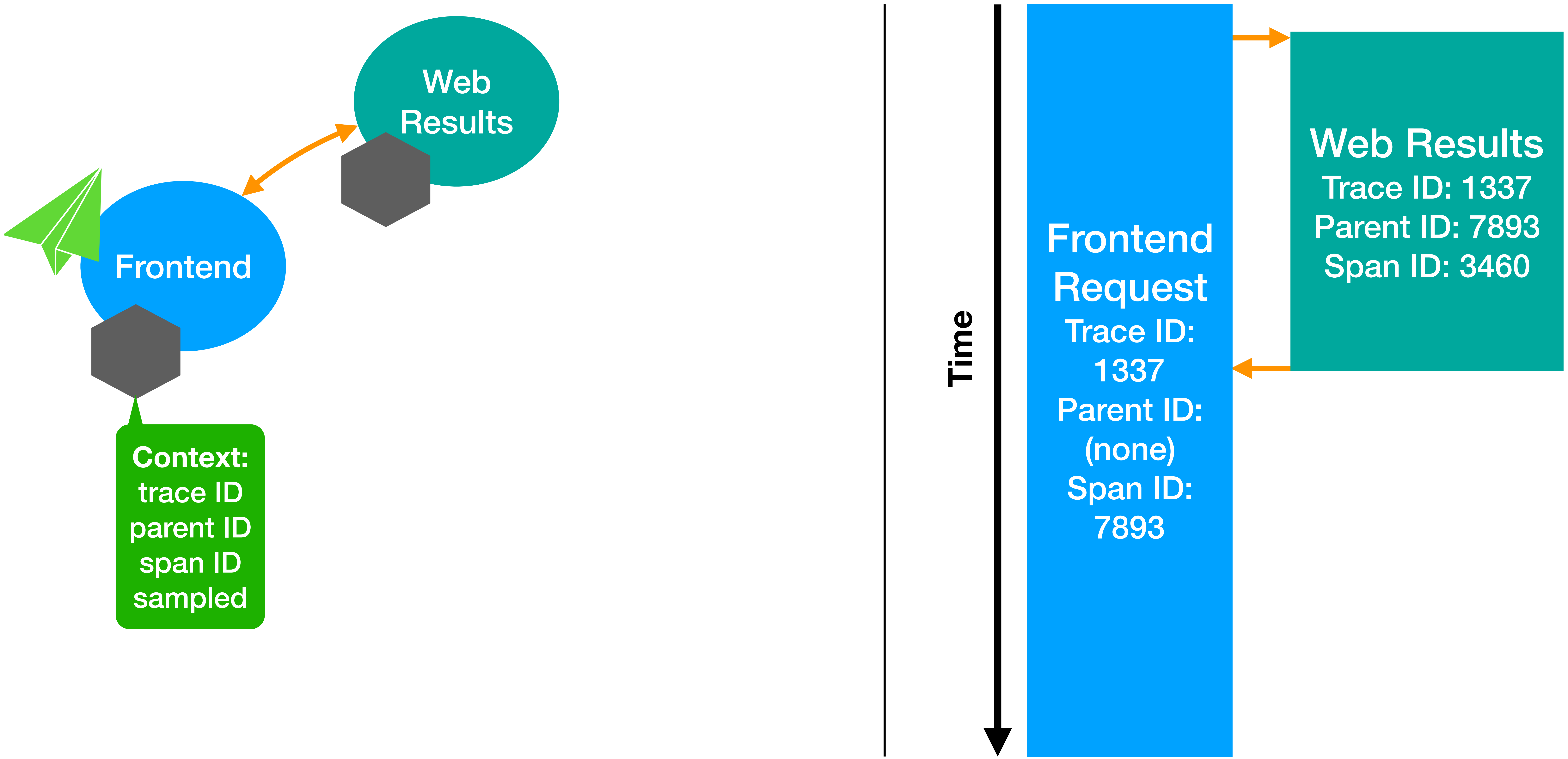
Traces and Spans



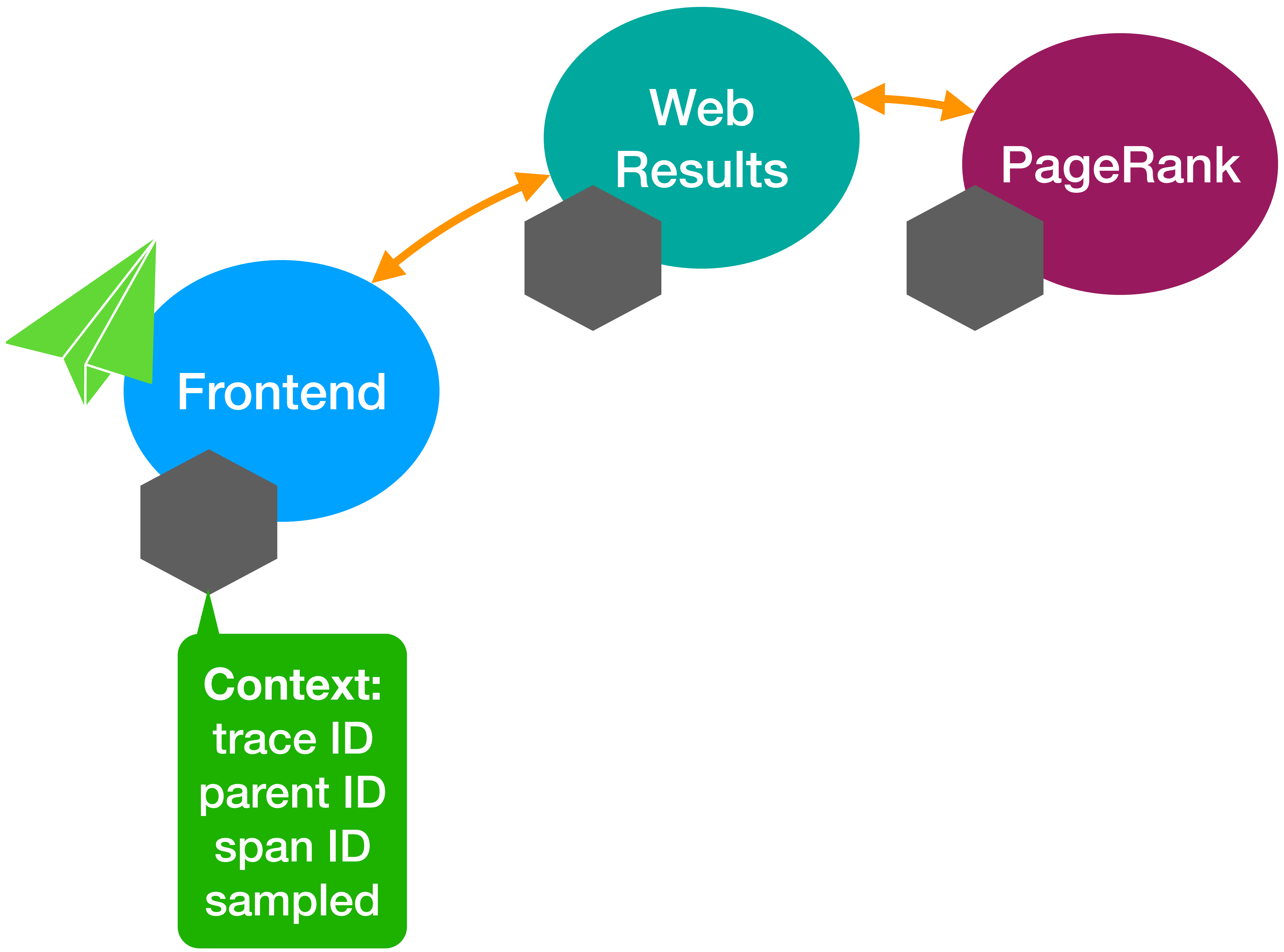
Traces and Spans

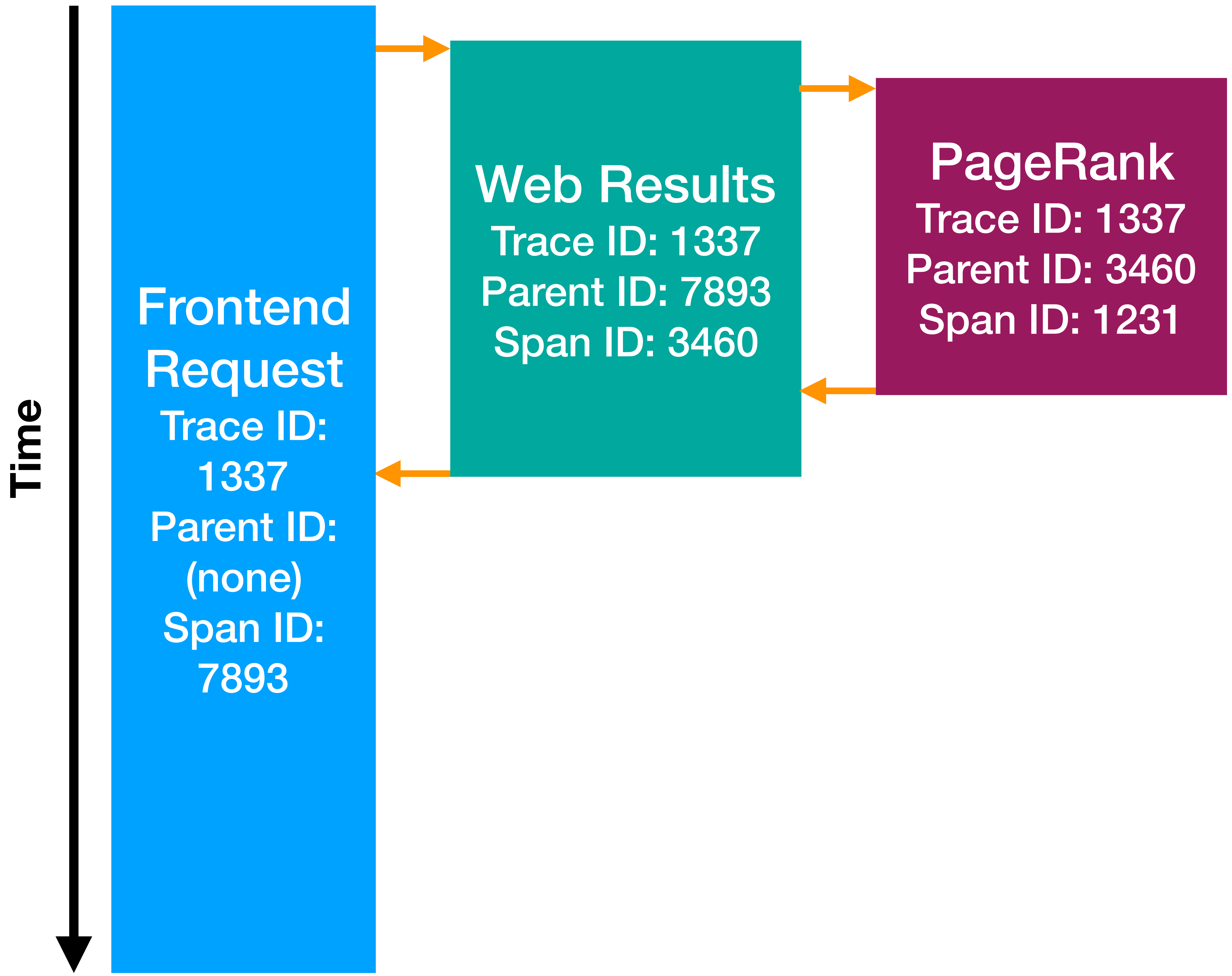


Traces and Spans

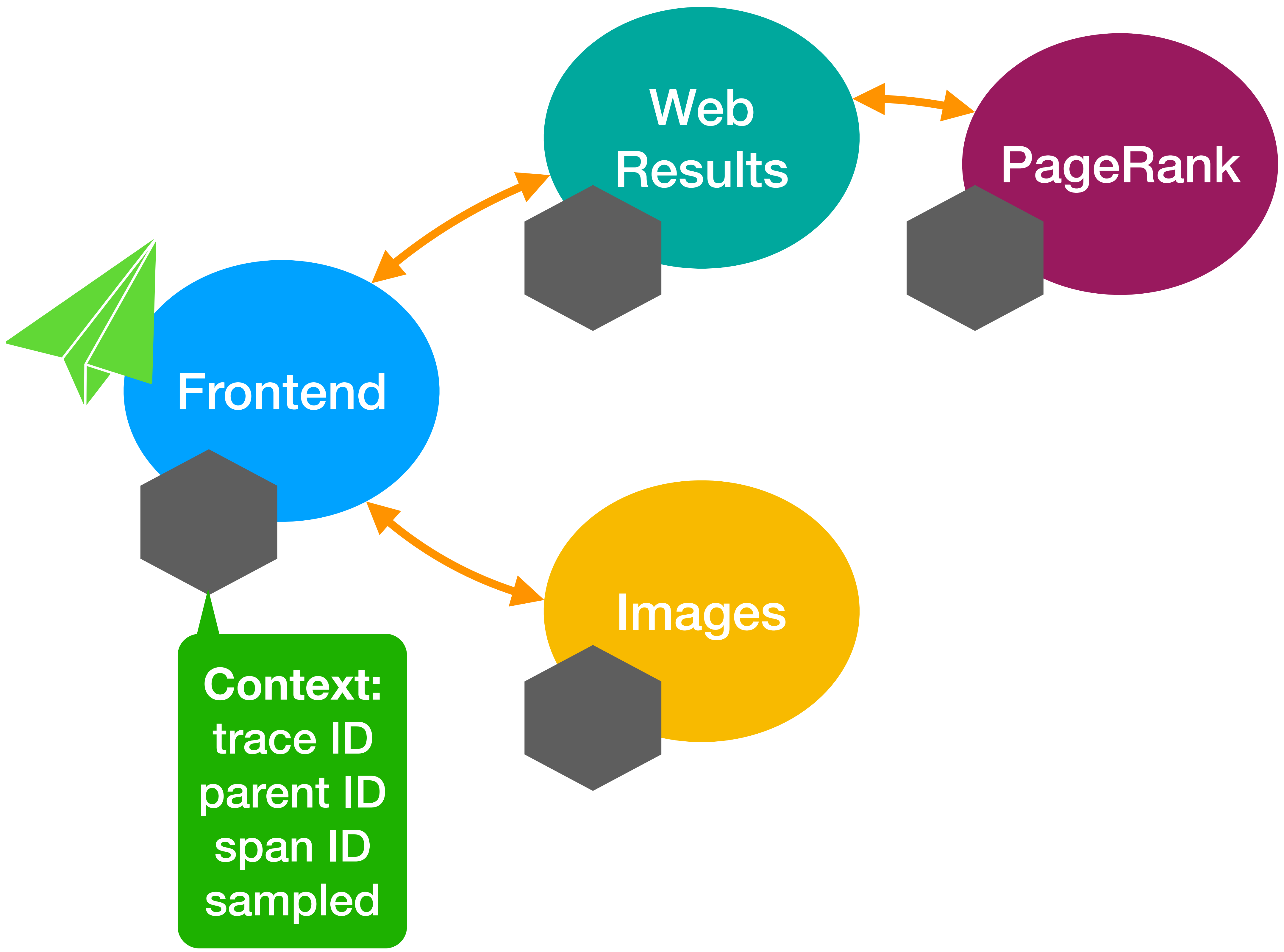


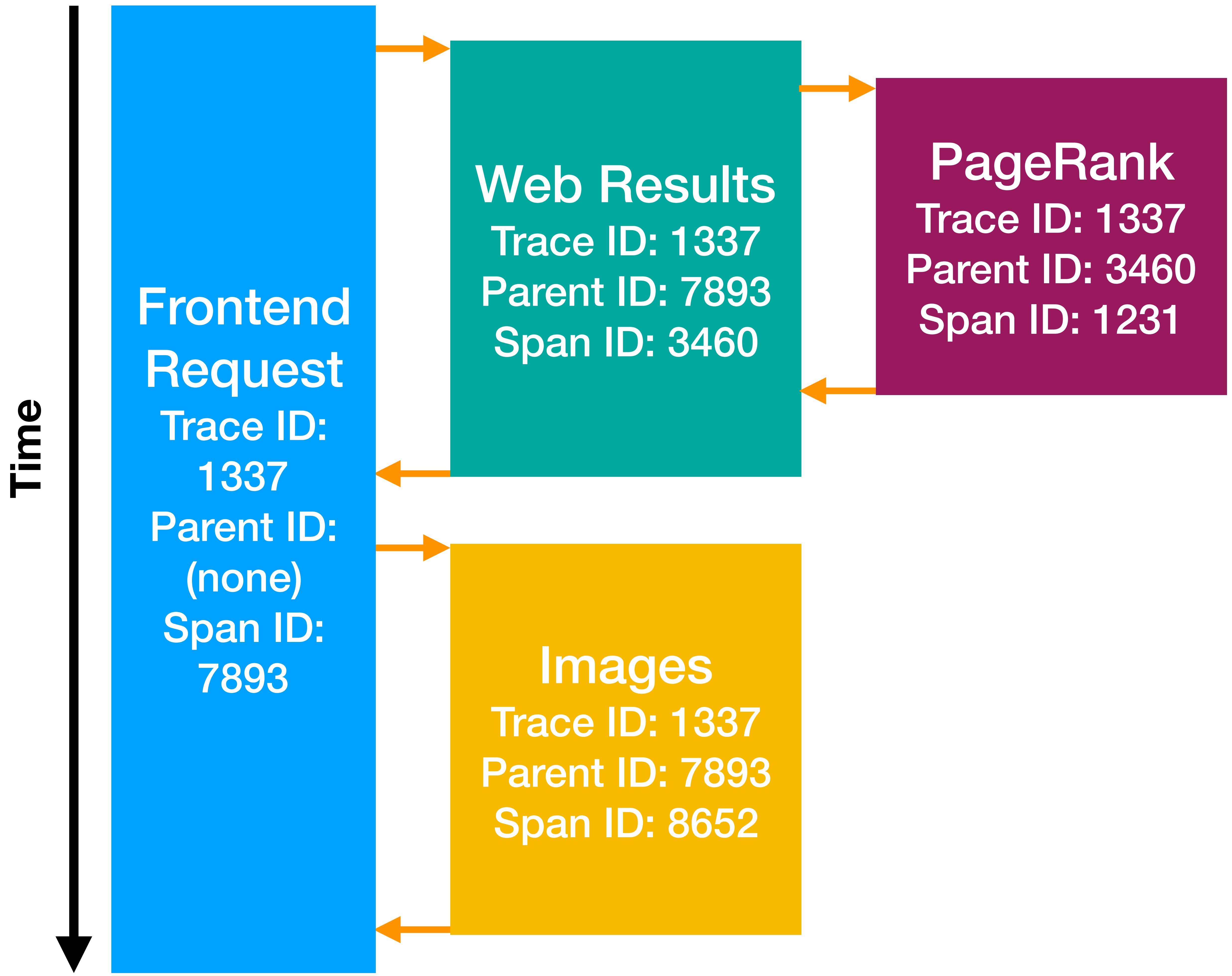
Traces and Spans



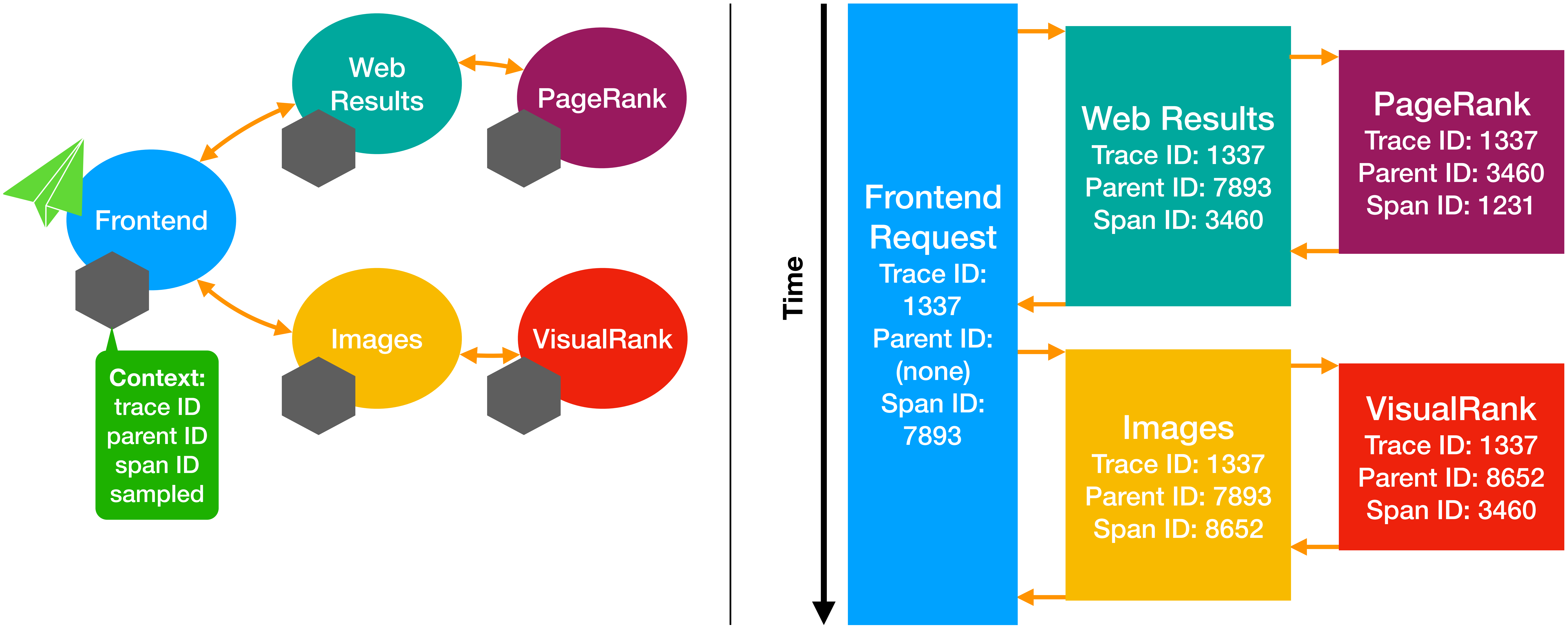


Traces and Spans

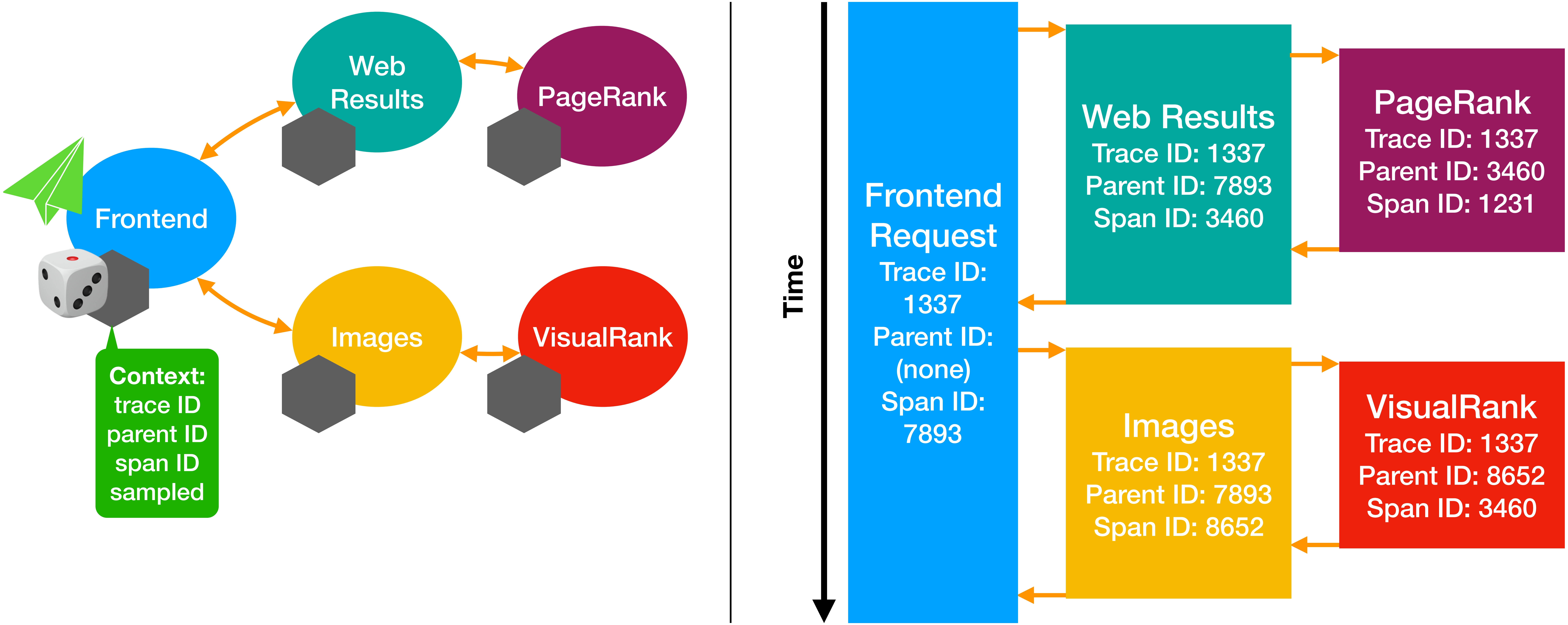




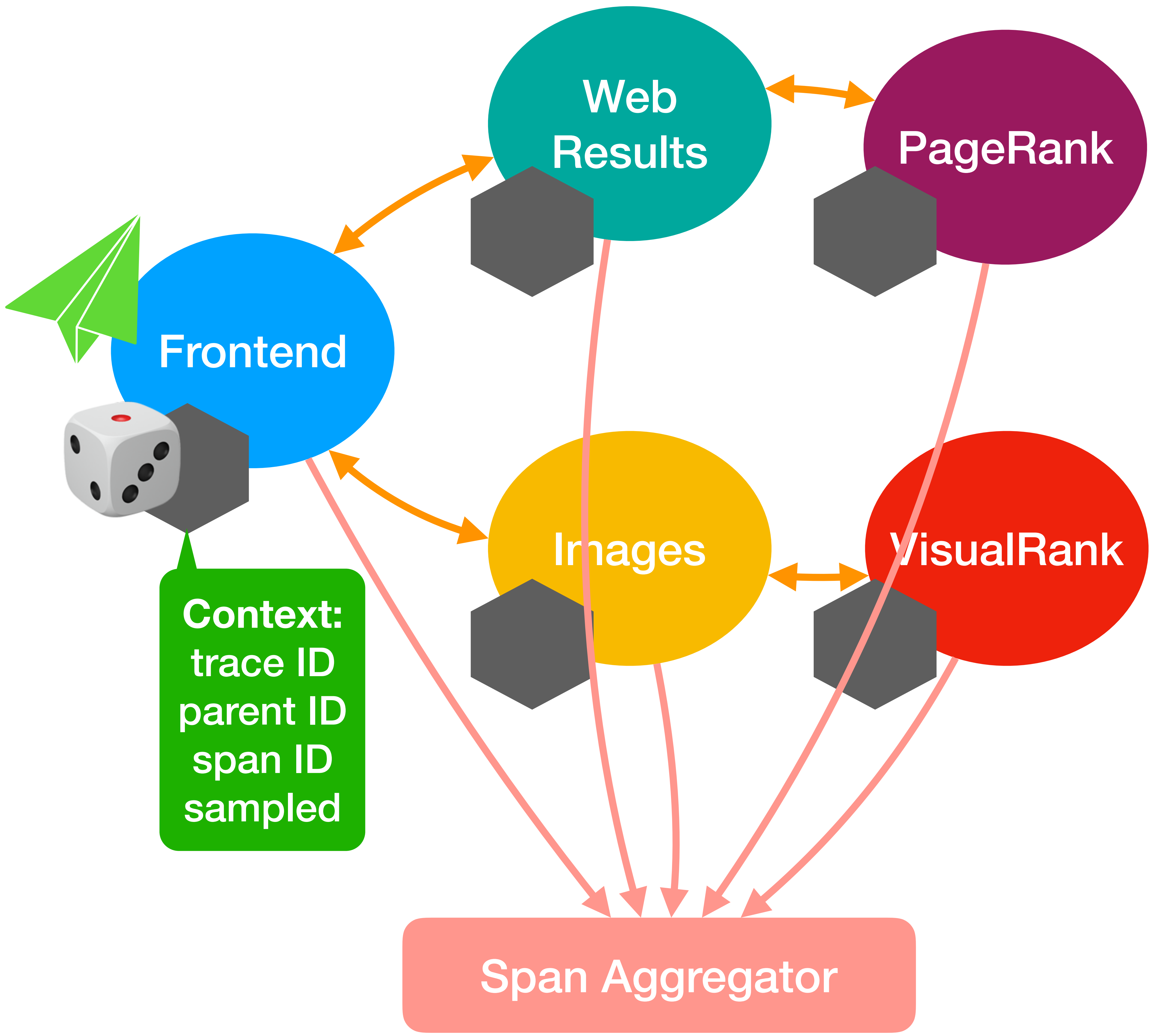
Traces and Spans

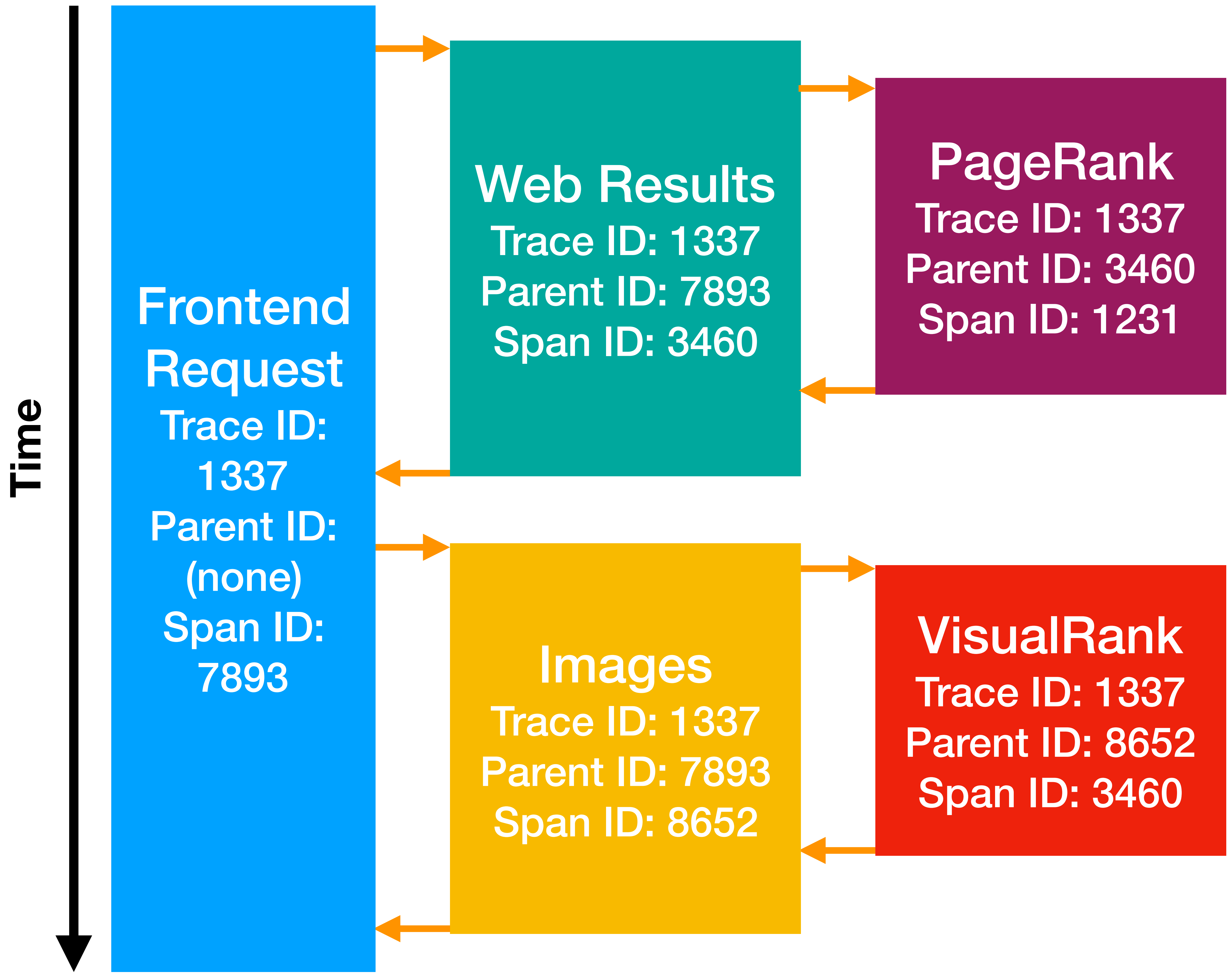


Traces and Spans

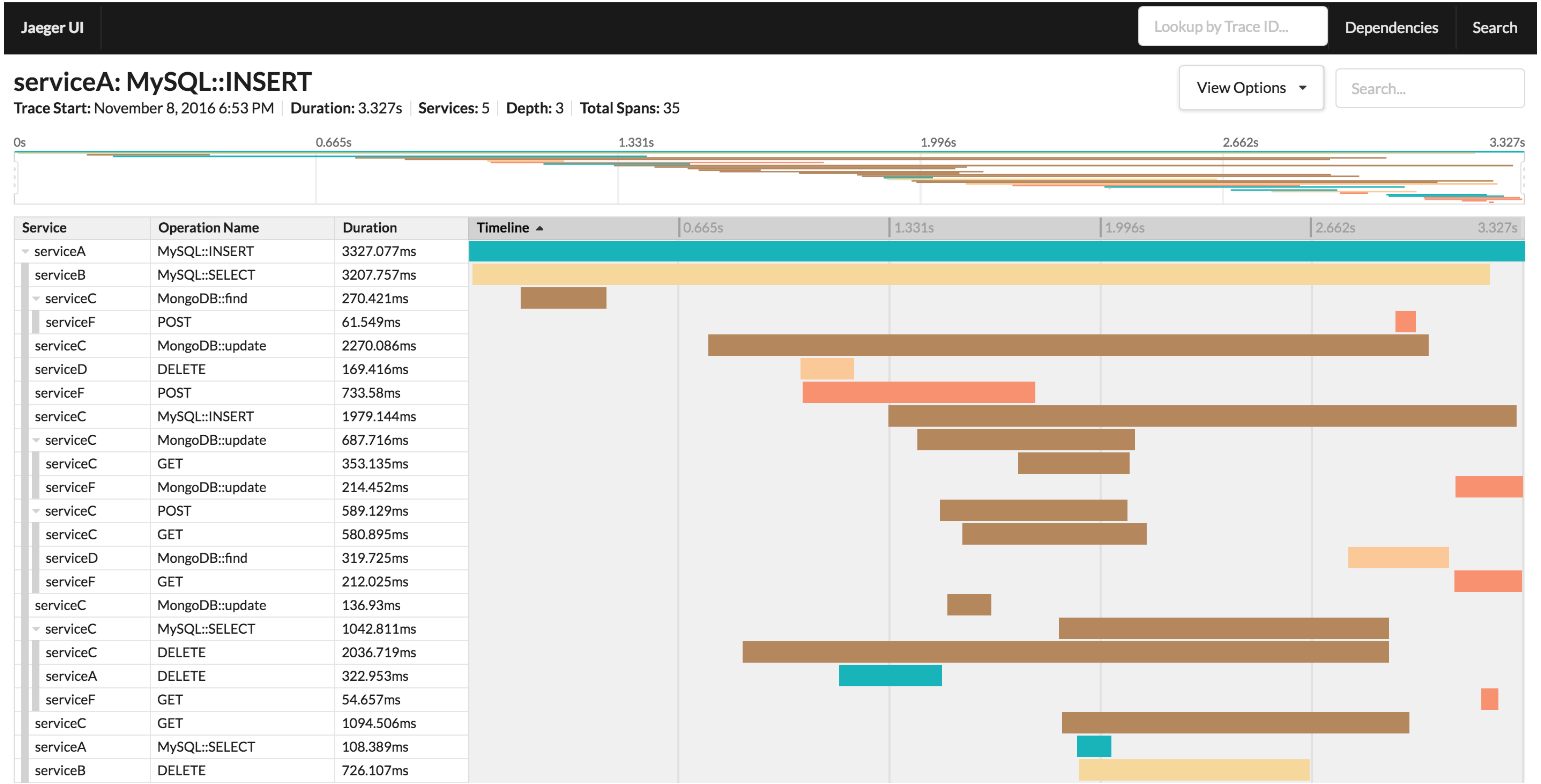


Traces and Spans



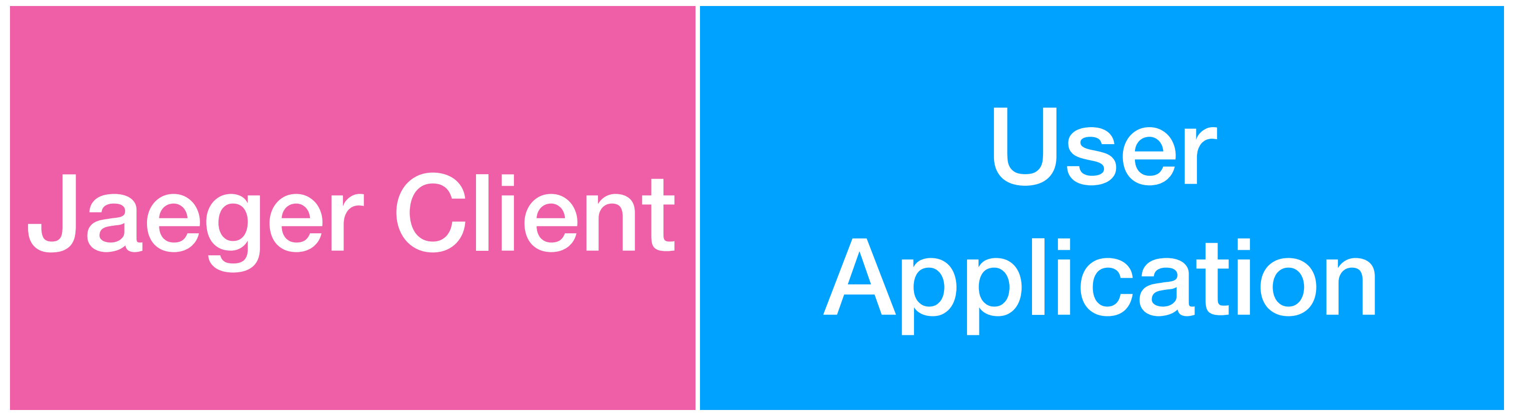


Jaeger

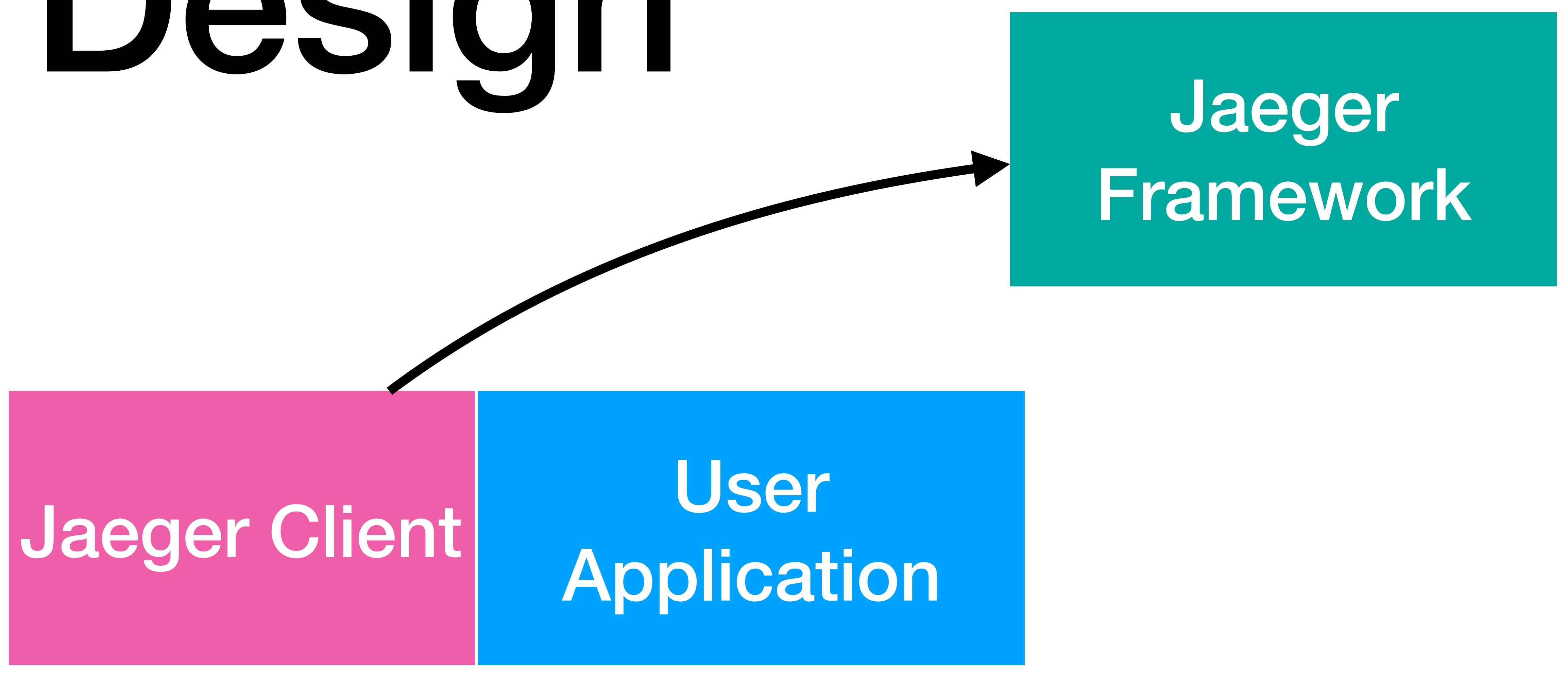


https://eng.uber.com/distributed-tracing/

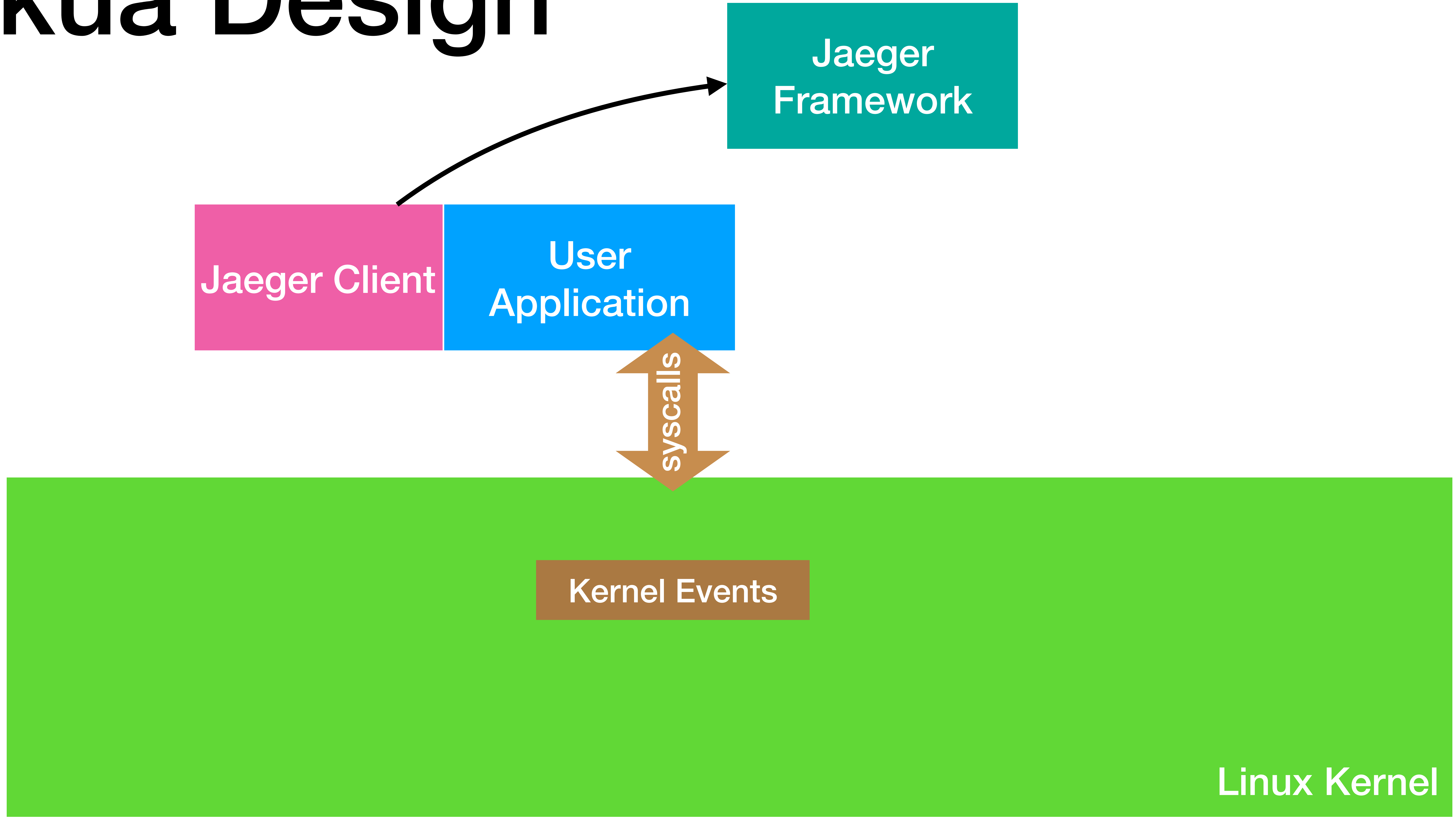
Skua Design



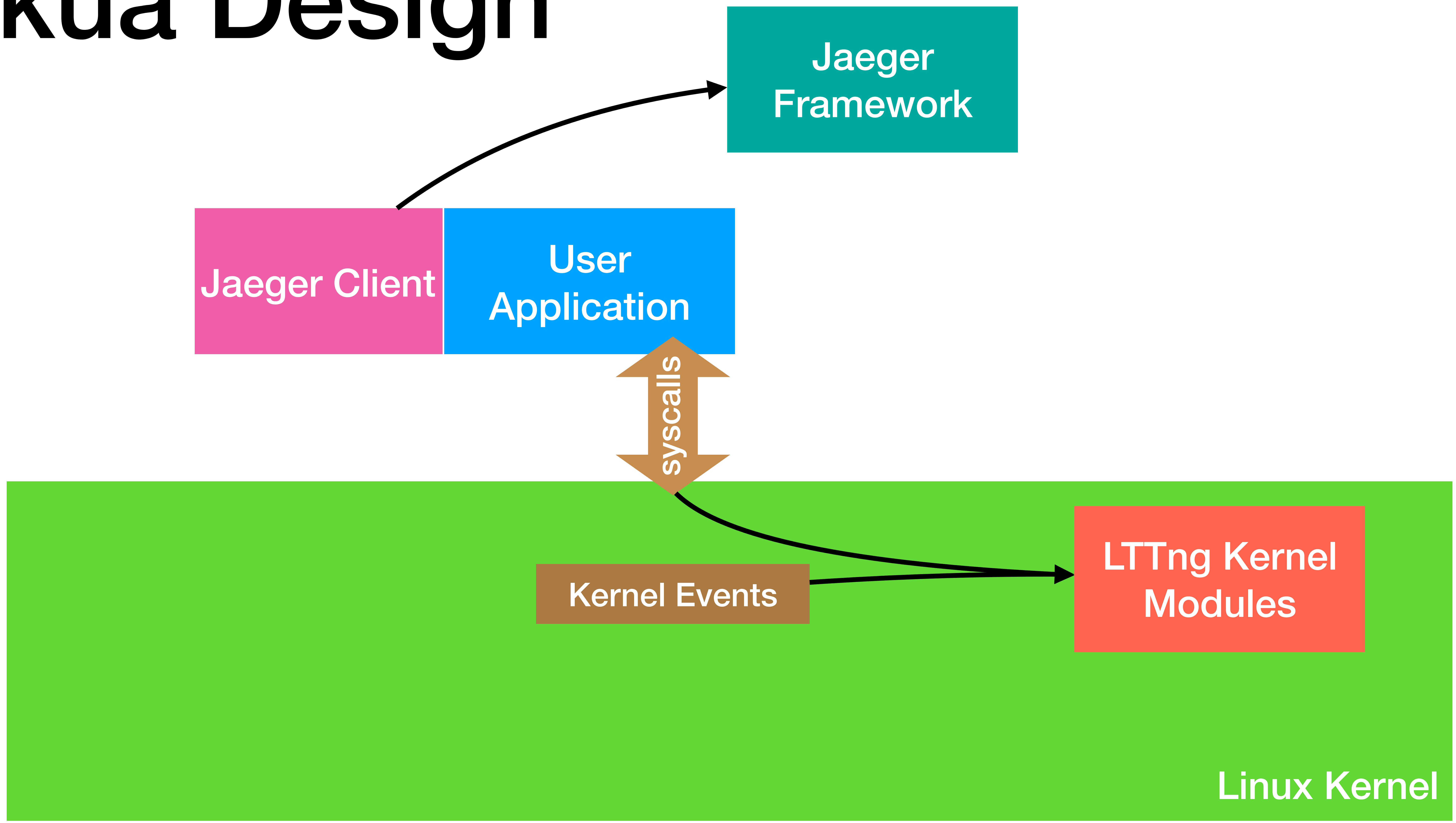
Skua Design



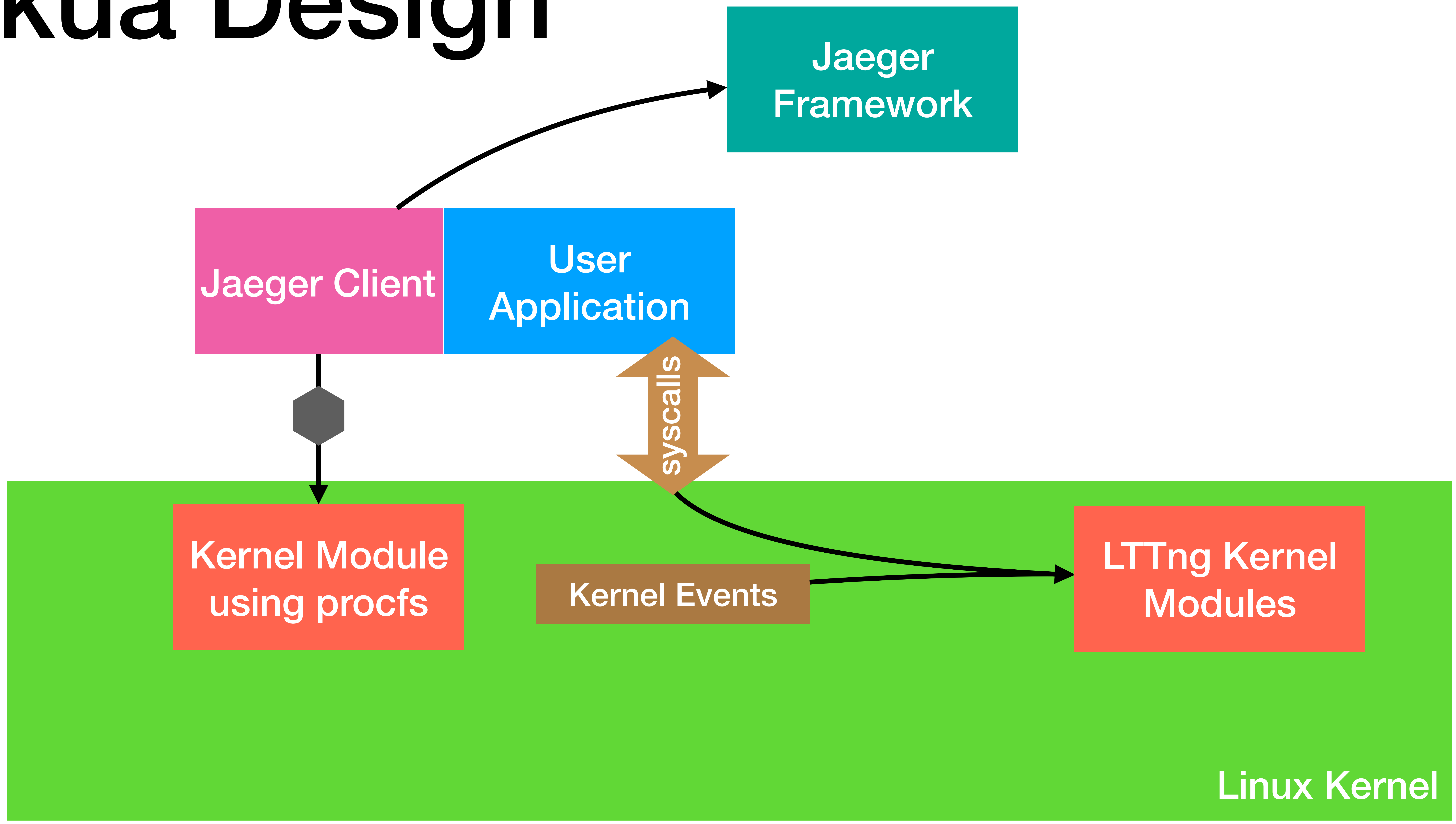
Skua Design


Jaeger Framework
Jaeger Client
User Application
syscalls
Kernel Events
Linux Kernel

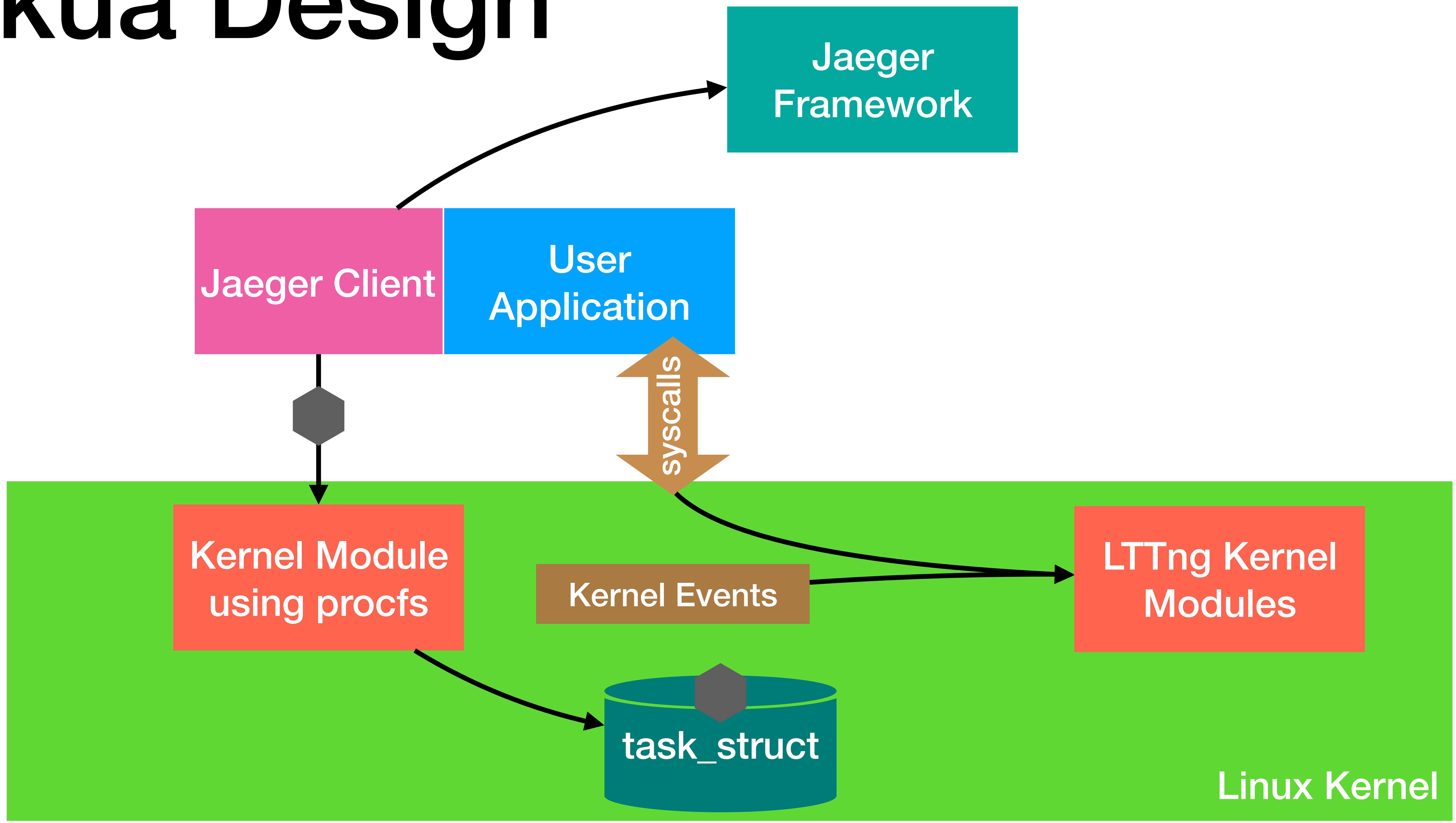
Skua Design


Jaeger Framework
Jaeger Client
User Application
syscalls
Kernel Events
LTTng Kernel Modules
Linux Kernel

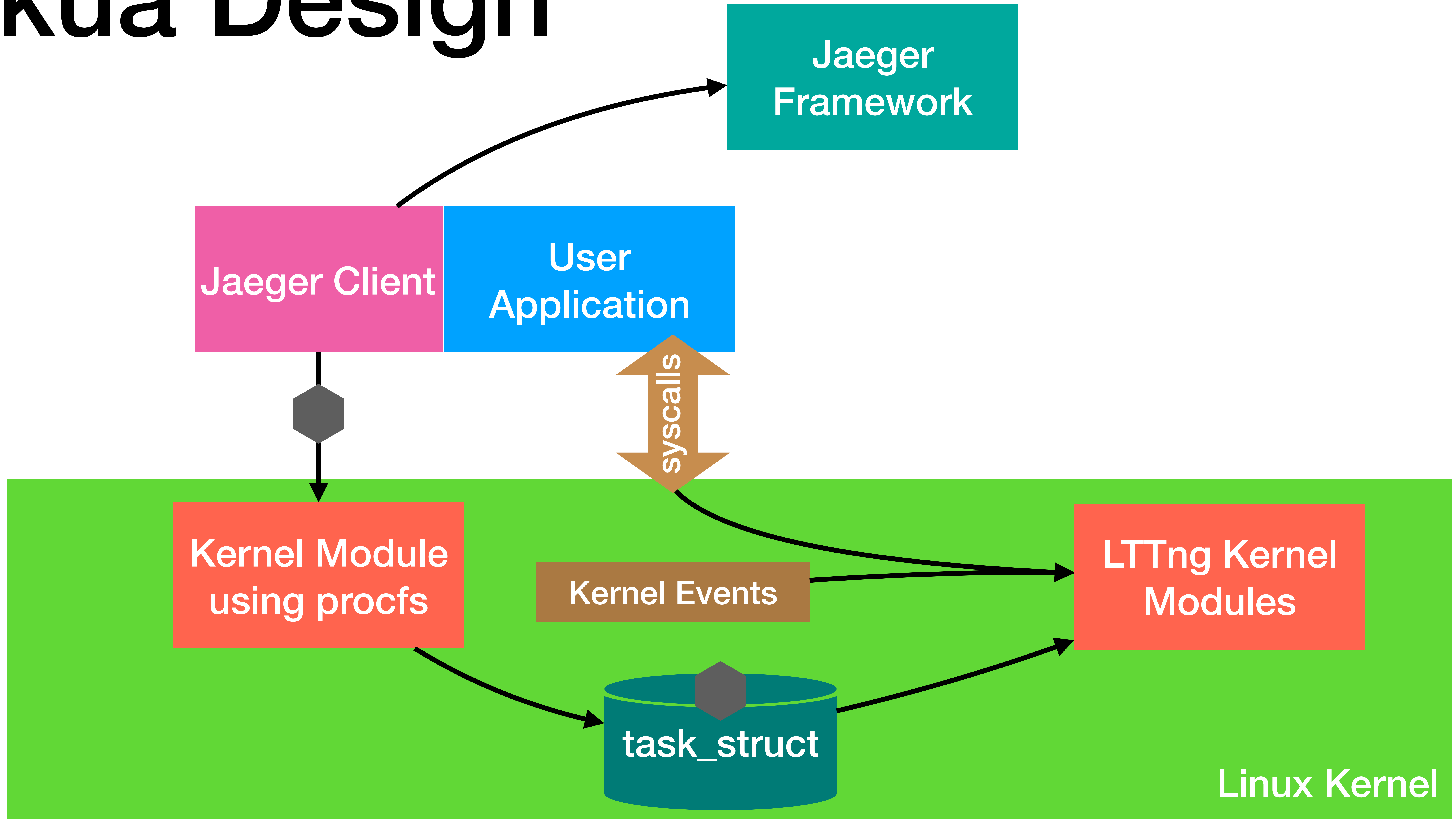
Skua Design


Jaeger Framework
Jaeger Client
User Application
syscalls
Kernel Module using procfs
Kernel Events
LTTng Kernel Modules
Linux Kernel

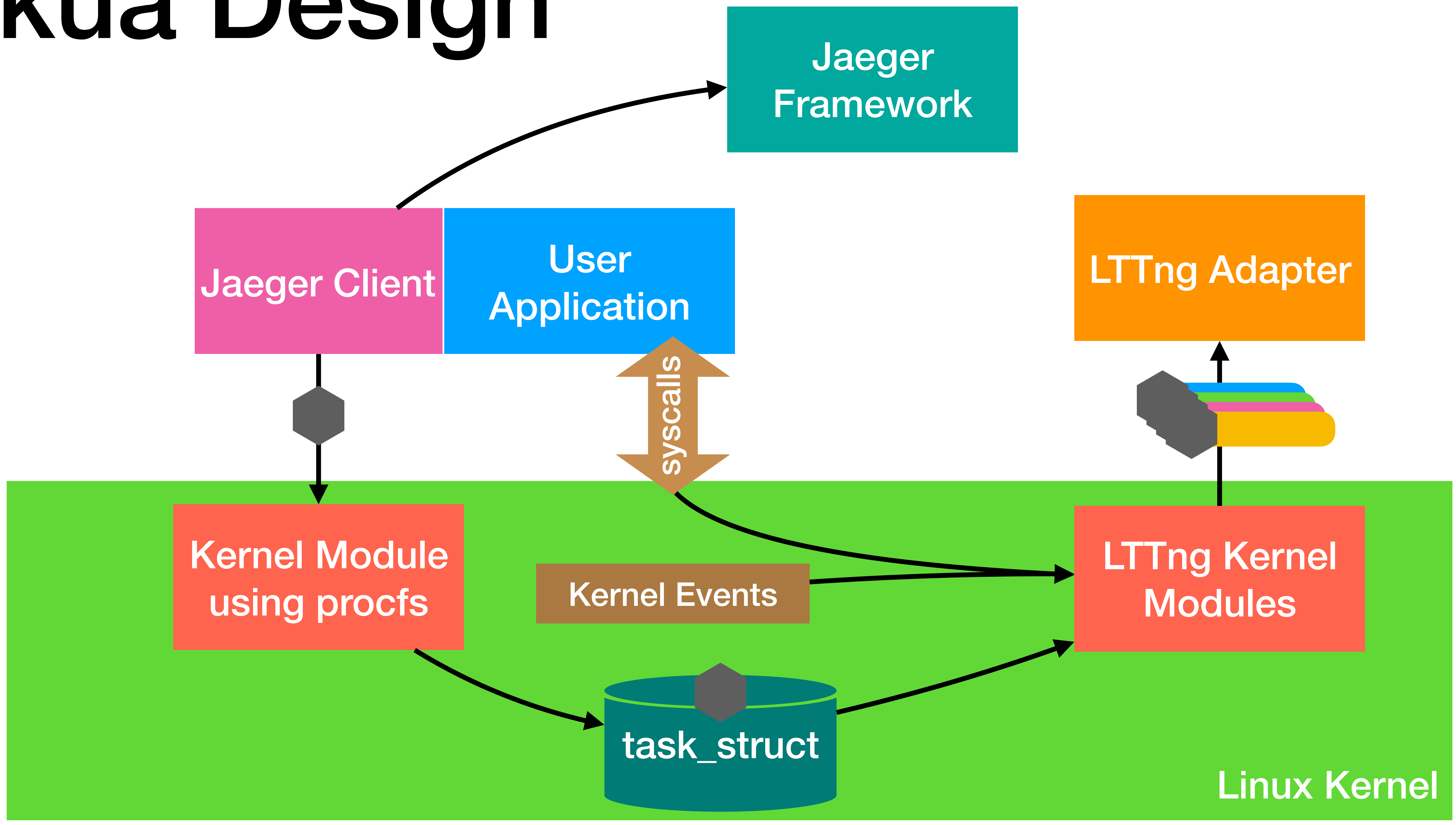
Skua Design


Jaeger Framework
Jaeger Client
User Application
syscalls
Kernel Module using procfs
Kernel Events
LTTng Kernel Modules
task_struct
Linux Kernel

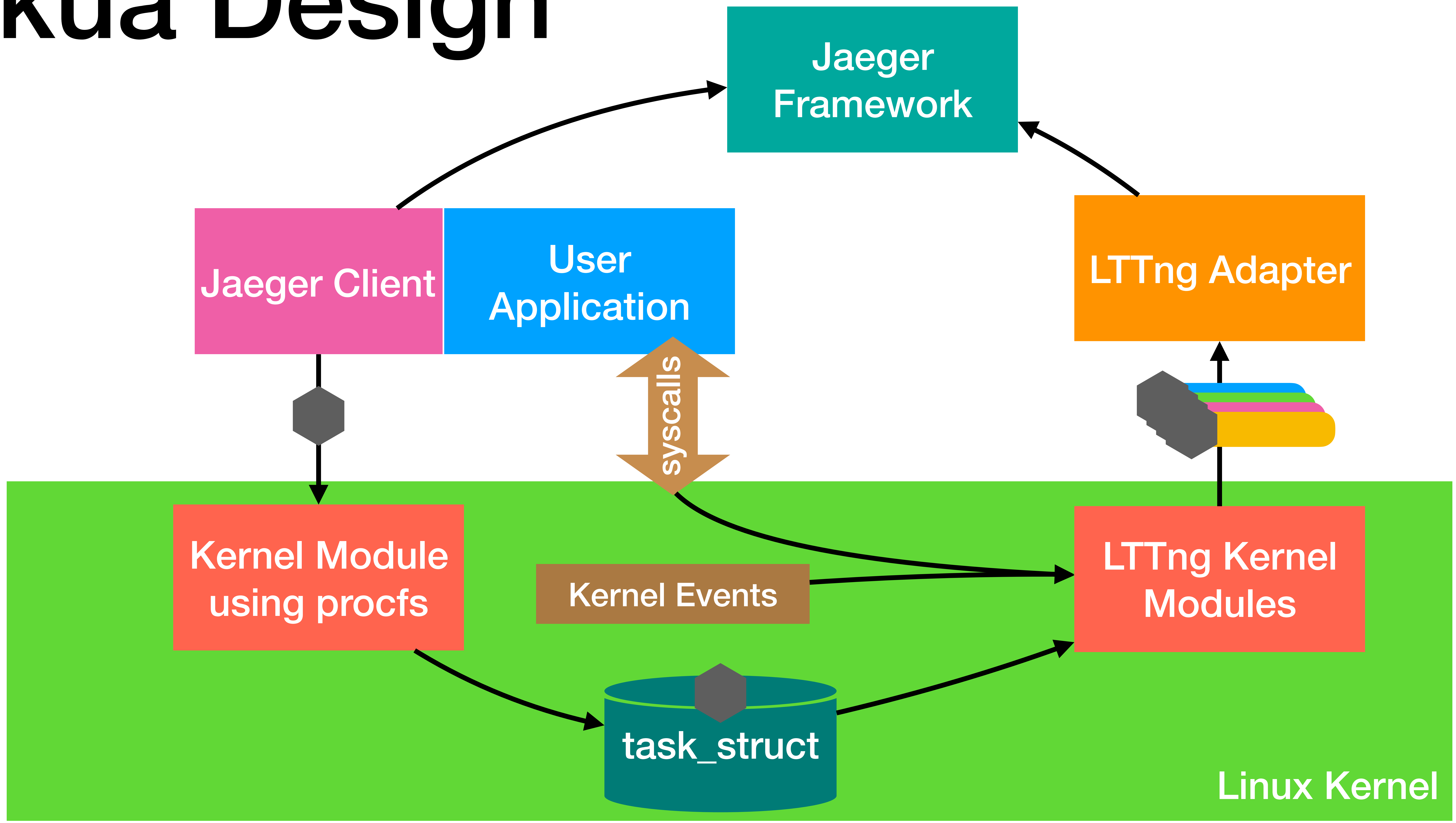
Skua Design


Jaeger Framework
Jaeger Client
User Application
syscalls
Kernel Module using procfs
Kernel Events
LTTng Kernel Modules
task_struct
Linux Kernel

Skua Design

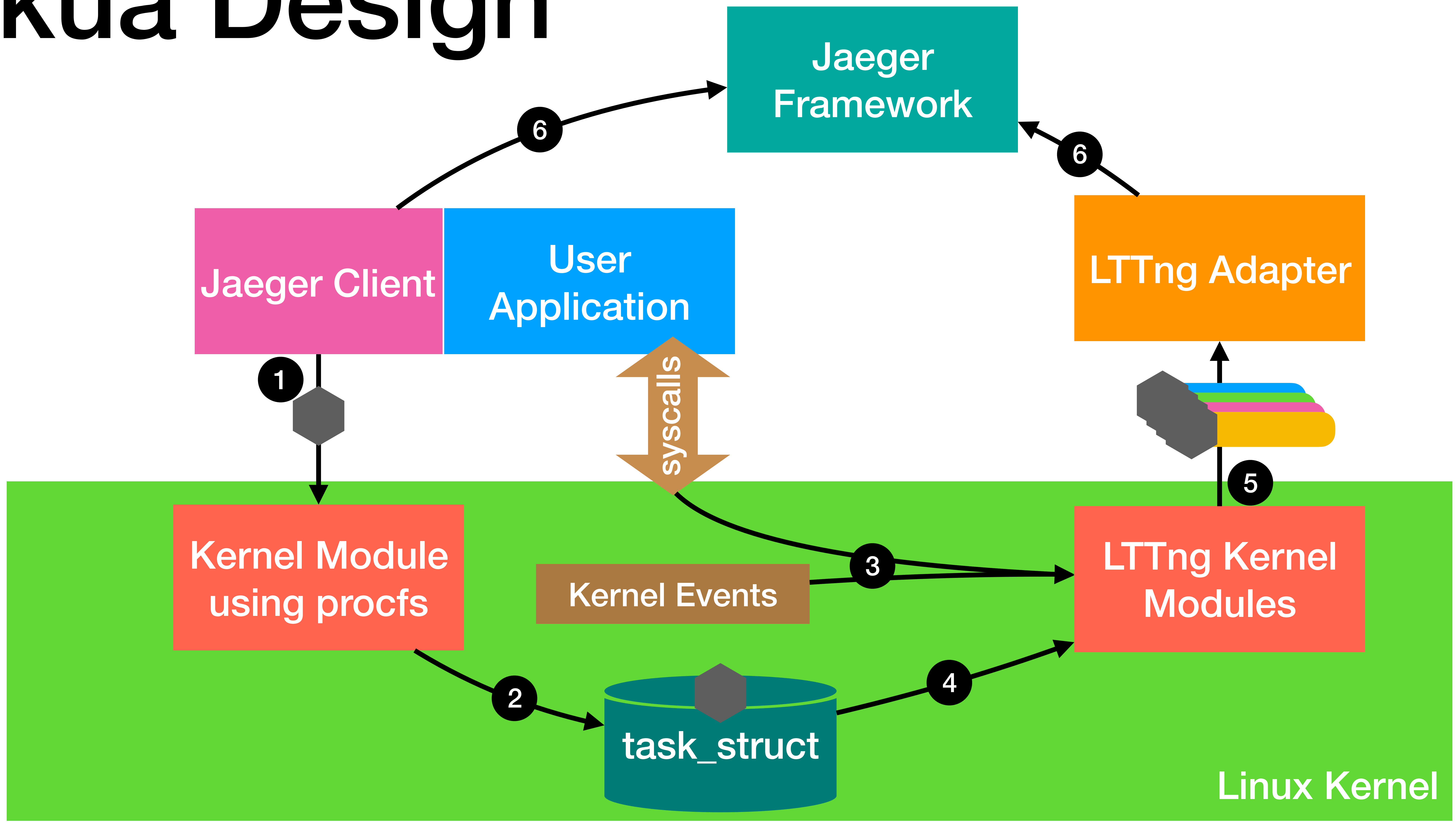

Jaeger Framework
Jaeger Client
User Application
LTTng Adapter
syscalls
Kernel Module using procfs
Kernel Events
LTTng Kernel Modules
task_struct
Linux Kernel

Skua Design



Jaeger Framework
Jaeger Client
User Application
LTTng Adapter
syscalls
Kernel Module using procfs
Kernel Events
LTTng Kernel Modules
task_struct
Linux Kernel

Skua Design


Jaeger Framework
6
6
Jaeger Client
User Application
LTTng Adapter
1
syscalls
5
Kernel Module using procfs
3
Kernel Events
LTTng Kernel Modules
2
4
task_struct
Linux Kernel

Skua Details

- Jaeger C++ client sends its context into the kernel 25 LOC

Skua Details

- Jaeger C++ client sends its context into the kernel 25 LOC
- Treats the Linux kernel as the next level of the span hierarchy
  - Each syscall is considered a span
  - Tracepoint events become span logs

Skua Details

- Jaeger C++ client sends its context into the kernel 25 LOC
- Treats the Linux kernel as the next level of the span hierarchy
  - Each syscall is considered a span
  - Tracepoint events become span logs
- LTTng kernel modules tag each span and log with the context information 80 LOC

Skua Details

- Jaeger C++ client sends its context into the kernel **25 LOC**
- Treats the Linux kernel as the next level of the span hierarchy
  - Each syscall is considered a span
  - Tracepoint events become span logs
- LTTng kernel modules tag each span and log with the context information **80 LOC**
- Custom adapter sends kernel data into the Jaeger **250 LOC**

Evaluation

Correctness

Setup

- Small C++ program
  - Spawns a few threads
  - Makes 10 different syscalls
- Verifies that Skua is correctly recording syscalls

Correctness

Setup

- Small C++ program
  - Spawns a few threads
  - Makes 10 different syscalls
- Verifies that Skua is correctly recording syscalls

Results

- Syscalls recorded in Jaeger as spans
- Misses a few syscalls
  - vDSO — `gettimeofday`
  - LTTng instrumentation
- Tracepoint events recorded properly as logs

Performance Overhead

- Created a small C++ Web server using uWebSockets

Performance Overhead

- Created a small C++ Web server using uWebSockets
- Used benchmarking tool autocannon

Performance Overhead

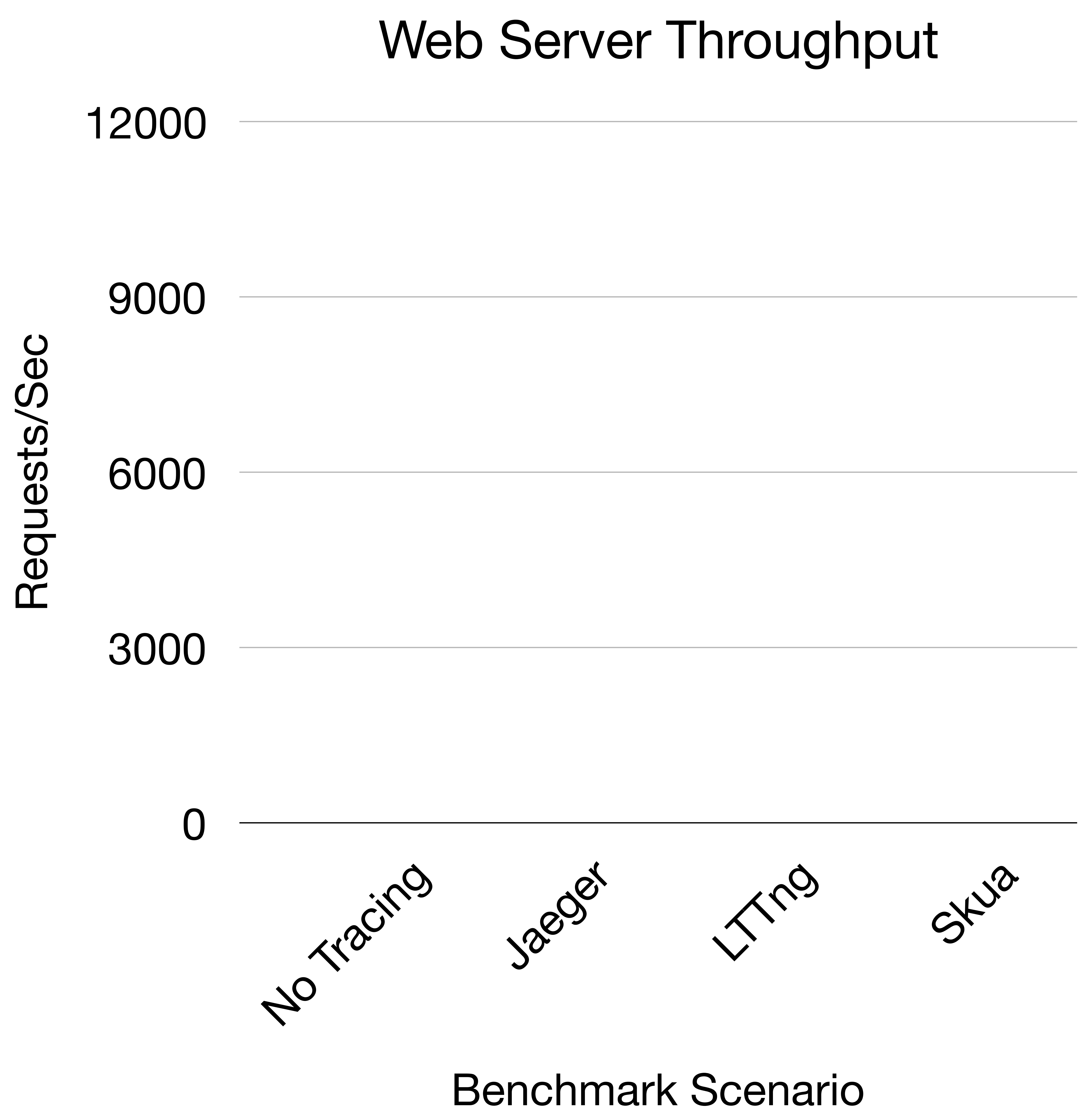
- Created a small C++ Web server using uWebSockets
- Used benchmarking tool autocannon
- Evaluated throughput and latency under different tracing scenarios

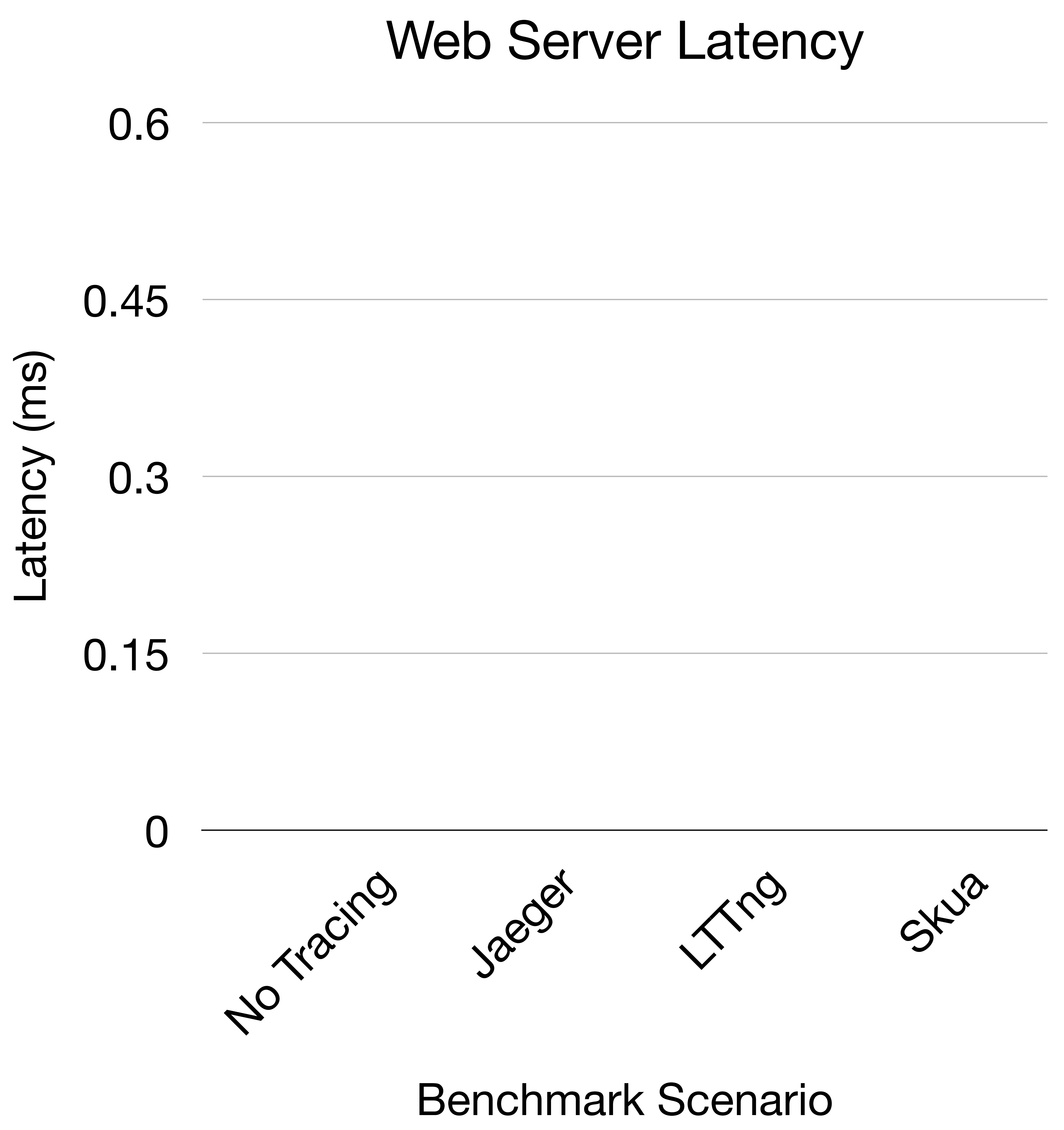
Performance Overhead

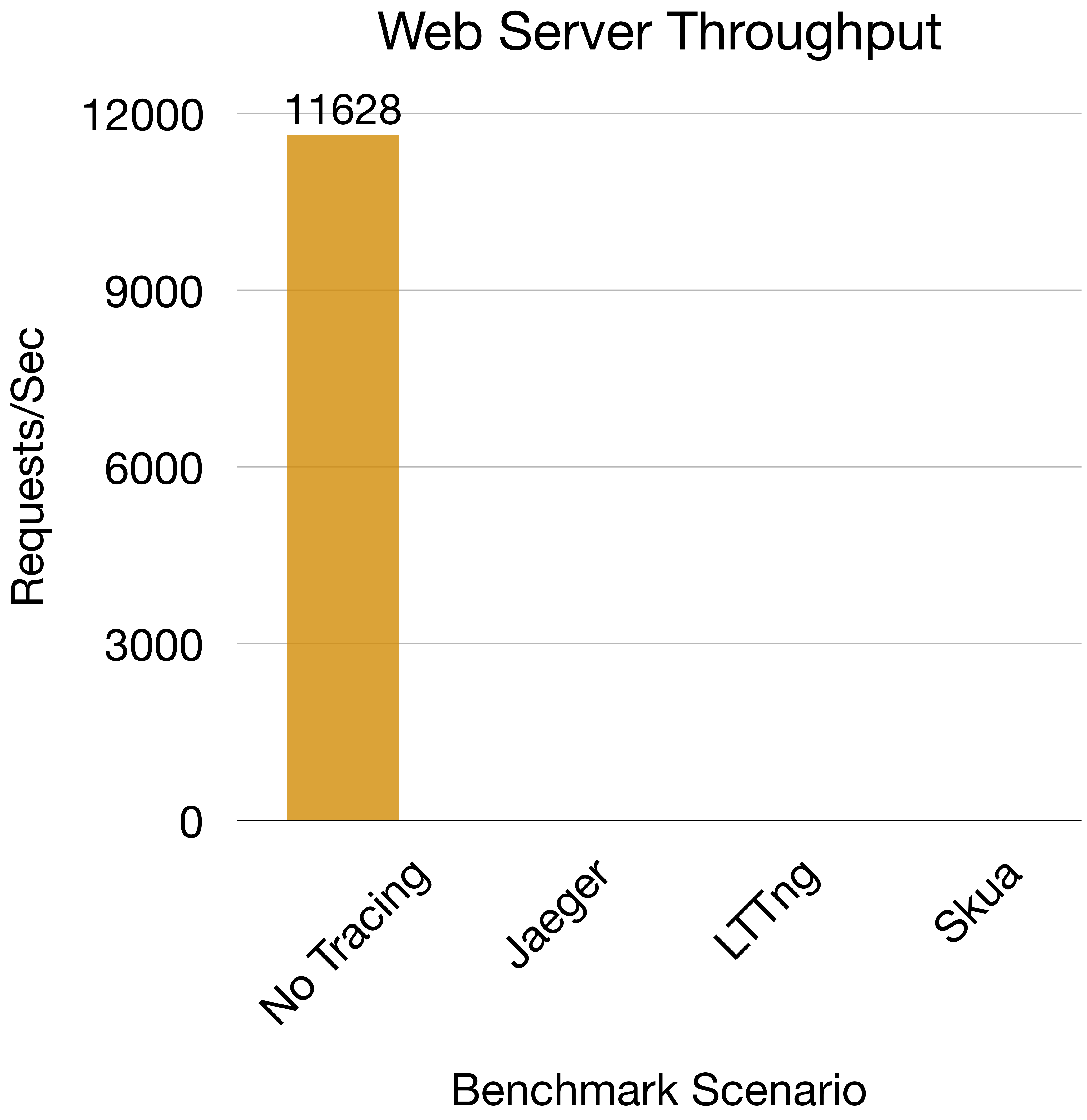
- Created a small C++ Web server using uWebSockets
- Used benchmarking tool autocannon
- Evaluated throughput and latency under different tracing scenarios
- 1,000,000 requests, 10 connections

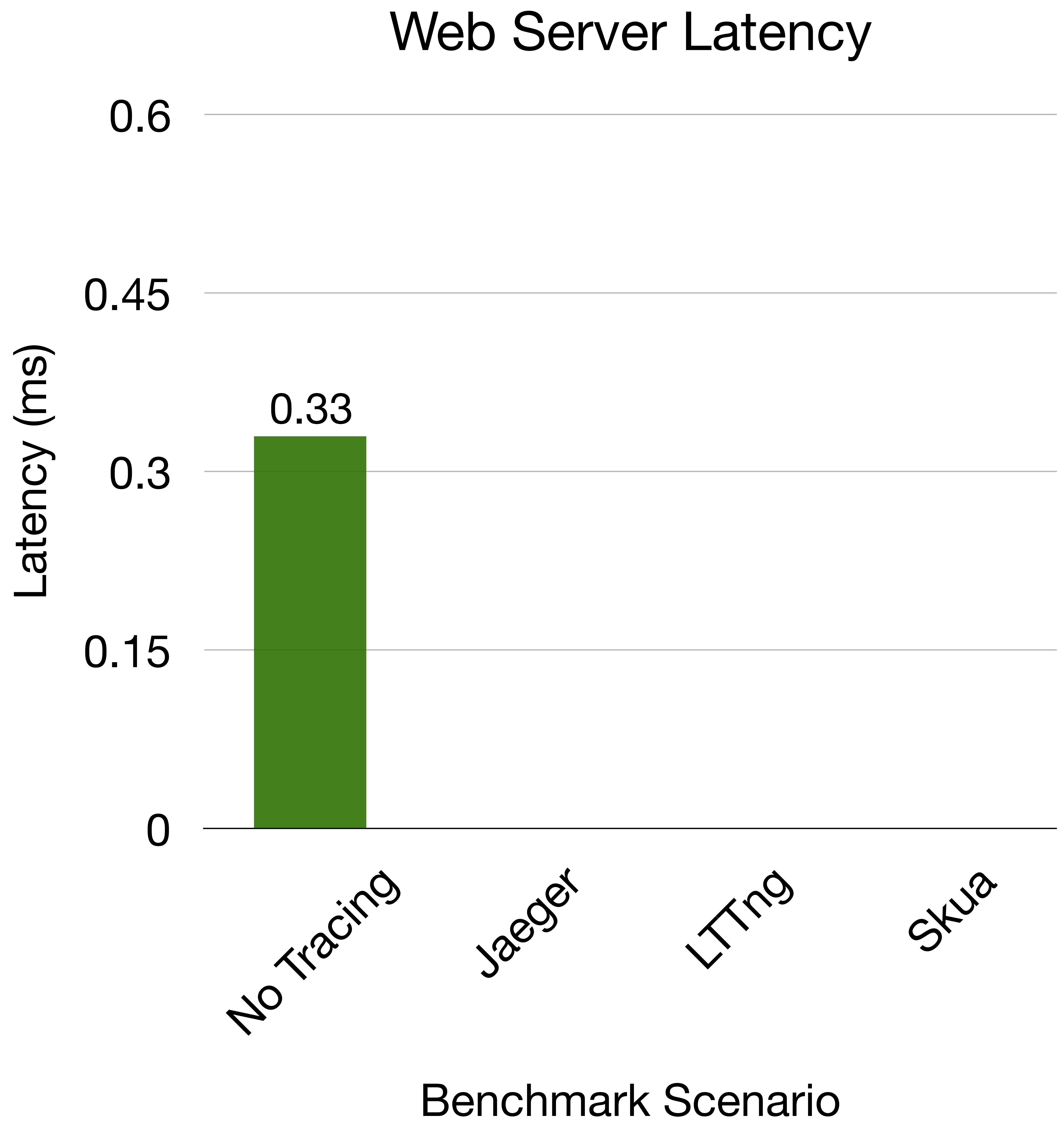
Performance Overhead

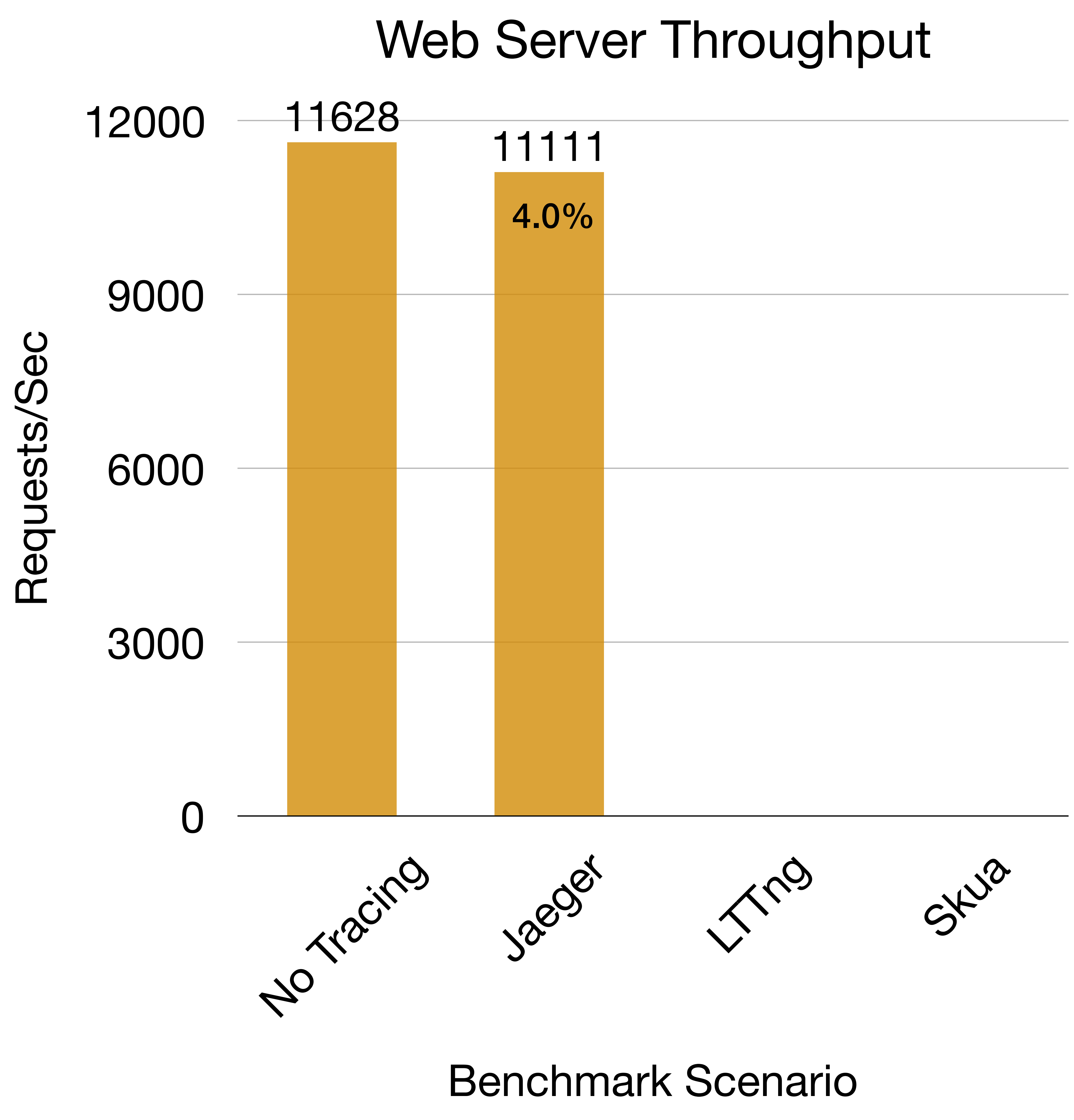
- Created a small C++ Web server using uWebSockets
- Used benchmarking tool autocannon
- Evaluated throughput and latency under different tracing scenarios
- 1,000,000 requests, 10 connections
- Traced 0.1% of requests

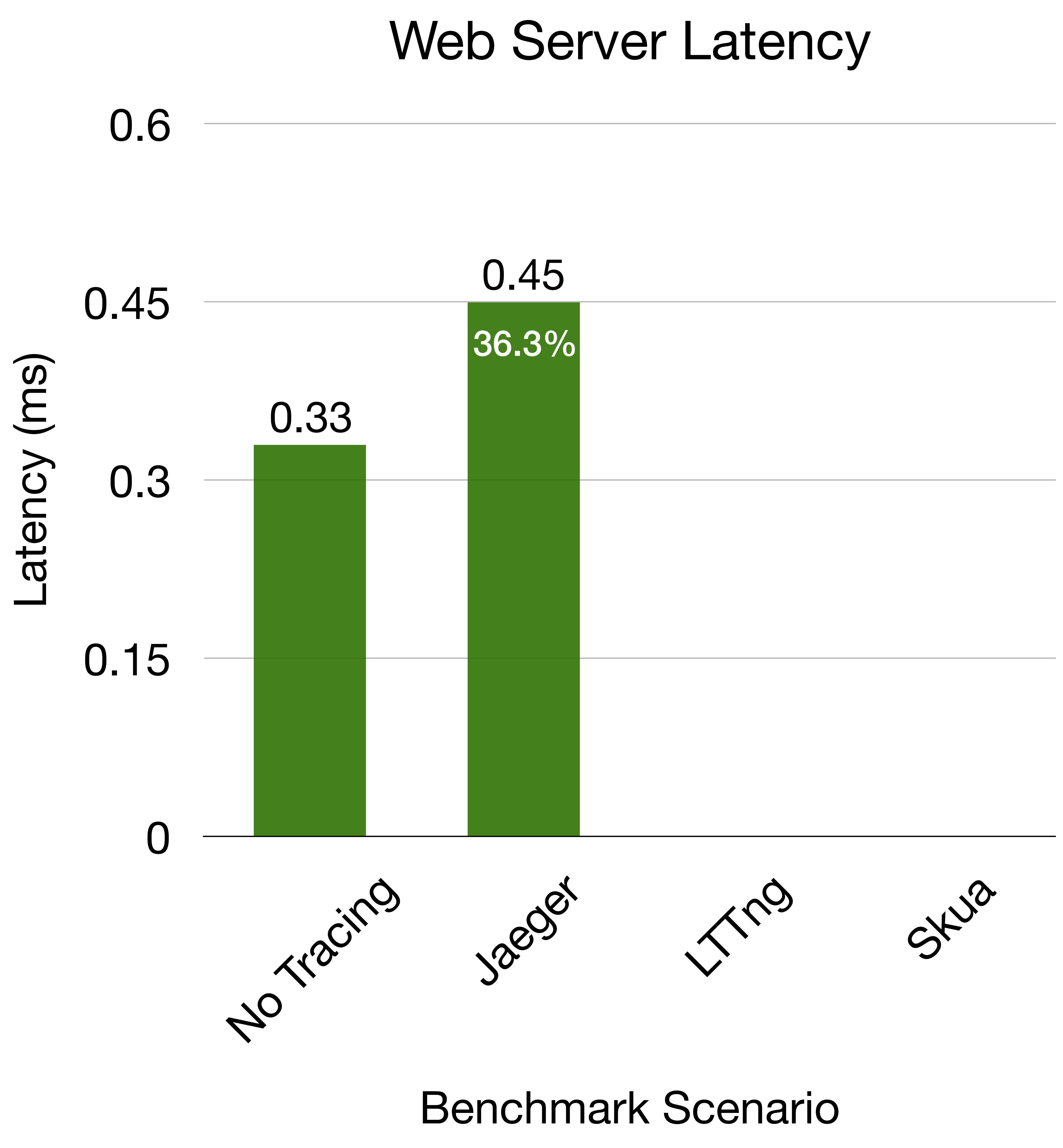

Web Server Throughput
12000
9000
6000
3000
0
Requests/Sec
No Tracing
Jaeger
LTTng
Skua
Benchmark Scenario

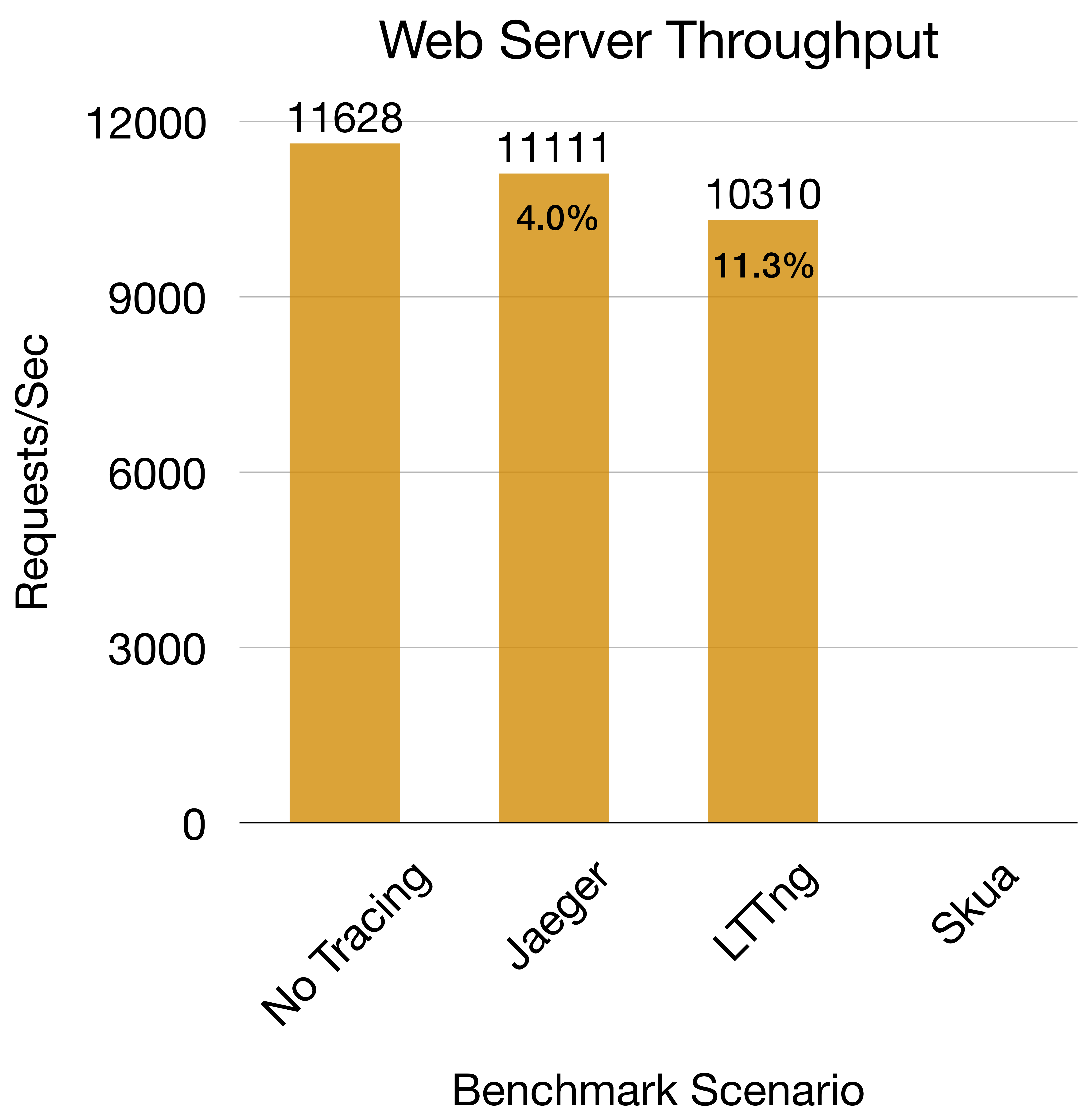

Web Server Latency
0.6
0.45
0.3
0.15
0
Latency (ms)
No Tracing
Jaeger
LTTng
Skua
Benchmark Scenario

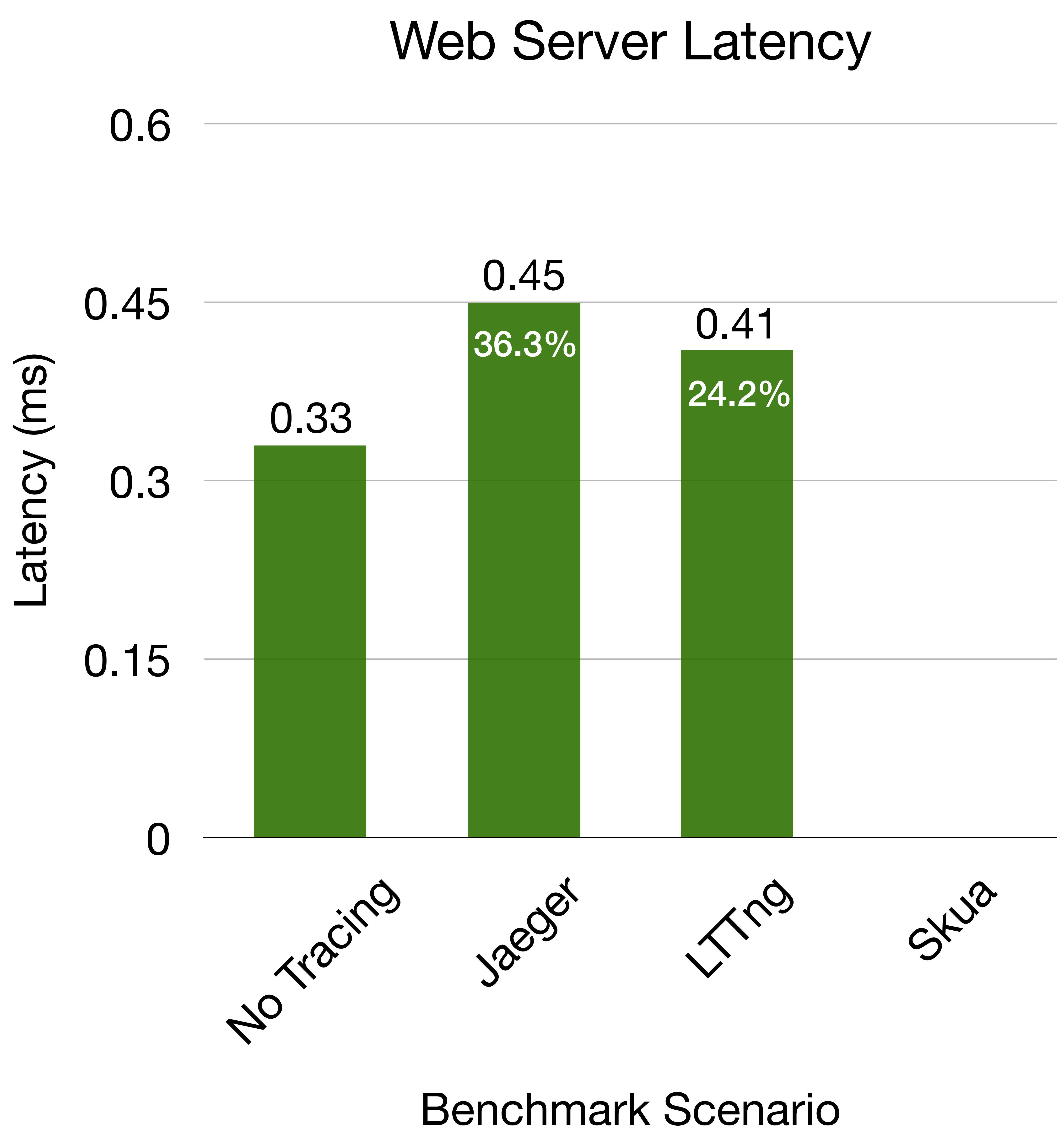

Web Server Throughput
11628
12000
9000
6000
3000
0
Requests/Sec
No Tracing
Jaeger
LTTng
Skua
Benchmark Scenario

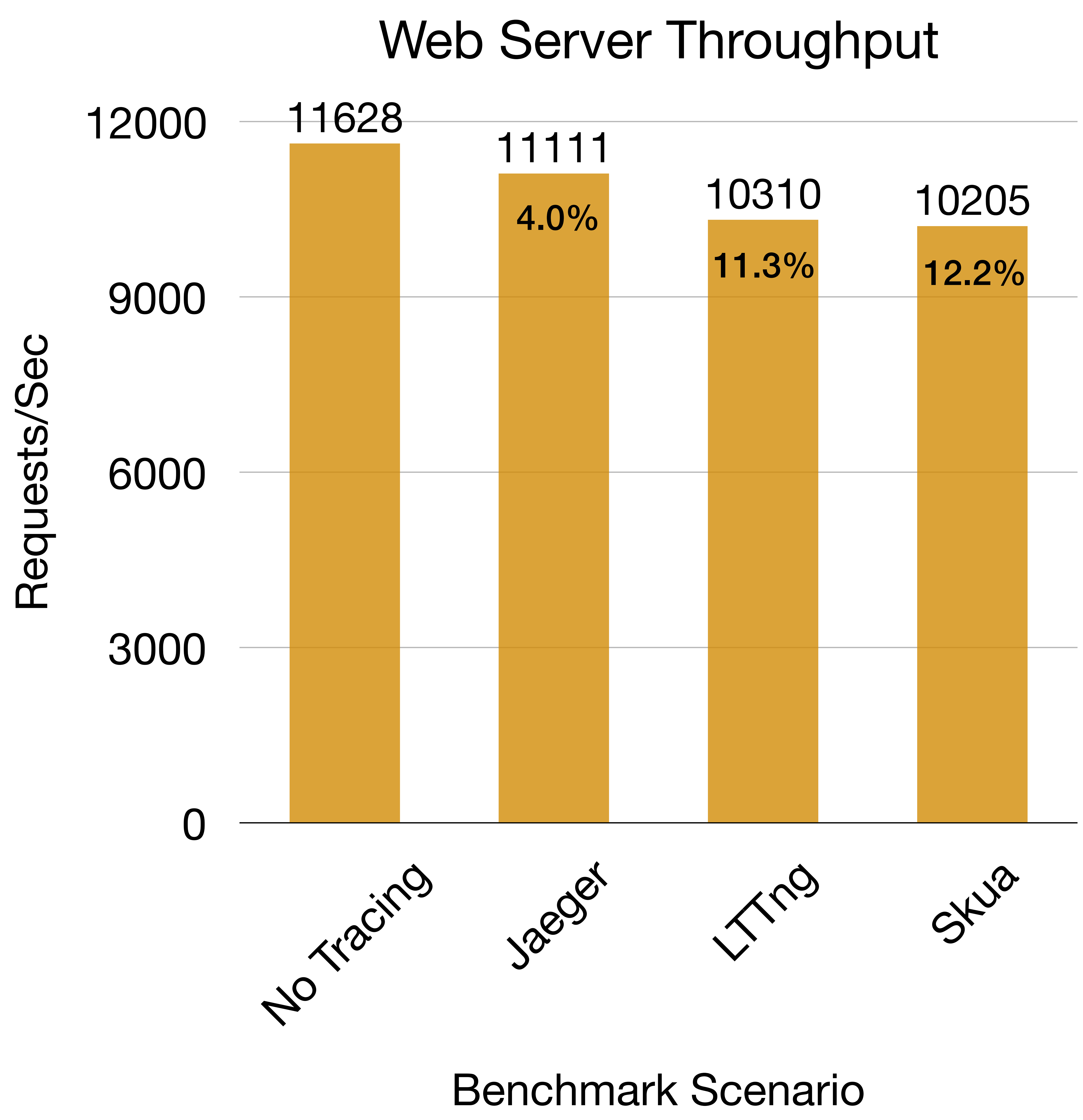

Web Server Latency
0.6
0.45
0.33
0.3
0.15
0
Latency (ms)
No Tracing
Jaeger
LTTng
Skua
Benchmark Scenario

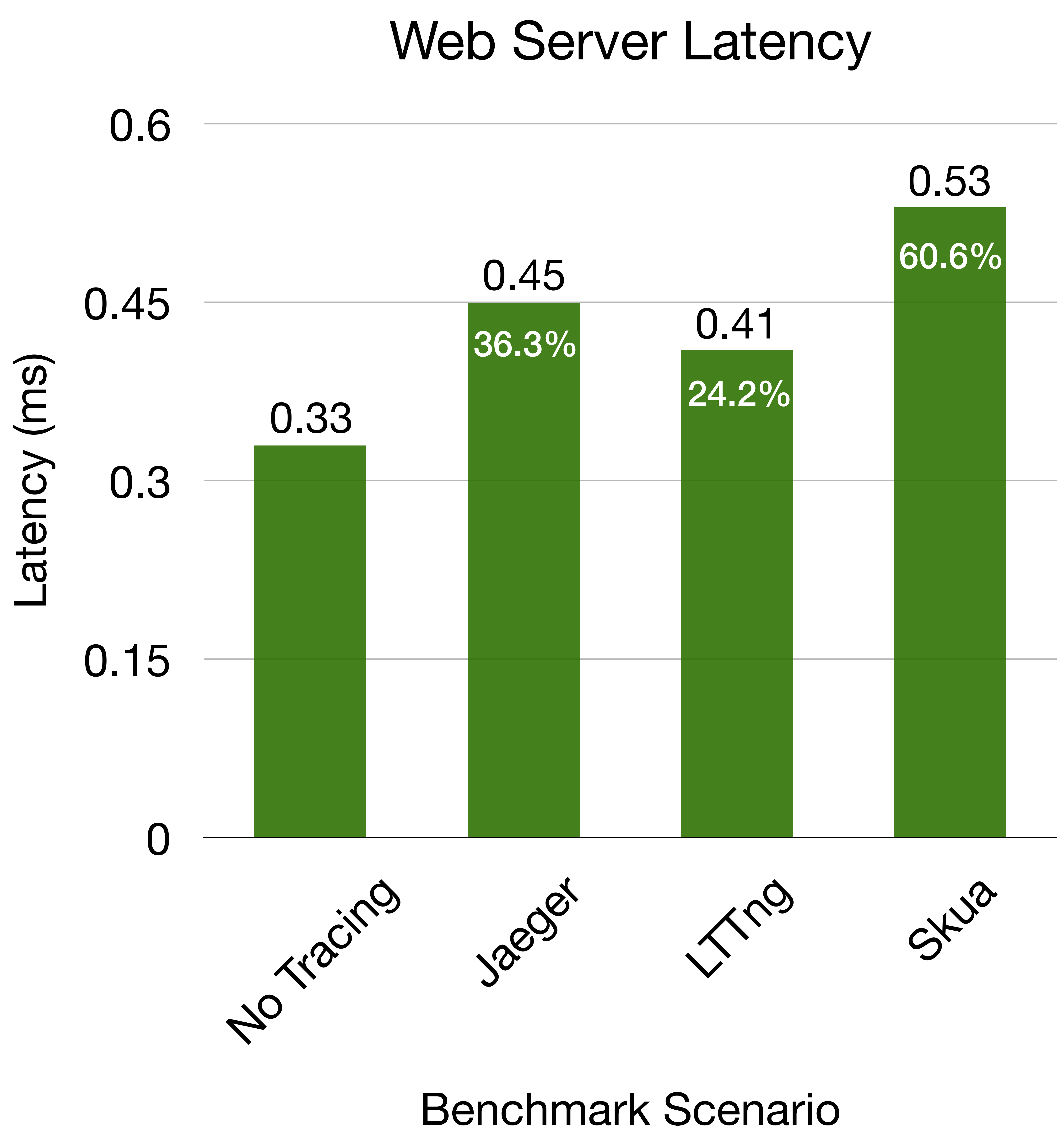

Web Server Throughput
12000
9000
6000
3000
0
Requests/Sec
11628
11111
4.0%
No Tracing
Jaeger
LTTng
Skua
Benchmark Scenario


Web Server Latency
0.6
0.45
0.3
0.15
0
Latency (ms)
0.33
0.45
36.3%
No Tracing
Jaeger
LTTng
Skua
Benchmark Scenario


Web Server Throughput
Requests/Sec
12000
9000
6000
3000
0
11628
11111
4.0%
10310
11.3%
No Tracing
Jaeger
LTTng
Skua
Benchmark Scenario


Web Server Latency
Latency (ms)
0.6
0.45
0.3
0.15
0
0.33
0.45
36.3%
0.41
24.2%
No Tracing
Jaeger
LTTng
Skua
Benchmark Scenario


Web Server Throughput
12000
9000
6000
3000
0
Requests/Sec
11628
11111
4.0%
10310
11.3%
10205
12.2%
No Tracing
Jaeger
LTTng
Skua
Benchmark Scenario


Web Server Latency
0.6
0.45
0.3
0.15
0
Latency (ms)
0.33
0.45
36.3%
0.41
24.2%
0.53
60.6%
No Tracing
Jaeger
LTTng
Skua
Benchmark Scenario

Conclusions

- Can use distributed tracing to monitor and debug complex distributed systems

Conclusions

- Can use distributed tracing to monitor and debug complex distributed systems
- Current open source distributed tracing frameworks miss kernel information

Conclusions

- Can use distributed tracing to monitor and debug complex distributed systems
- Current open source distributed tracing frameworks miss kernel information
- Skua integrates LTTng kernel data with Jaeger tracing

Conclusions

- Can use distributed tracing to monitor and debug complex distributed systems
- Current open source distributed tracing frameworks miss kernel information
- Skua integrates LTTng kernel data with Jaeger tracing
- Skua has some impact on throughput and latency

Conclusions

- Can use distributed tracing to monitor and debug complex distributed systems
- Current open source distributed tracing frameworks miss kernel information
- Skua integrates LTTng kernel data with Jaeger tracing
- Skua has some impact on throughput and latency
- “[12.2%] ought to be [good] enough for anybody”
  —Bill Gates, paraphrased

Conclusions

- Can use distributed tracing to monitor and debug complex distributed systems
- Current open source distributed tracing frameworks miss kernel information
- Skua integrates LTTng kernel data with Jaeger tracing
- Skua has some impact on throughput and latency
- “[12.2%] ought to be [good] enough for anybody”
  —Bill Gates, paraphrased

https://github.com/SkuaTracing

Acknowledgements

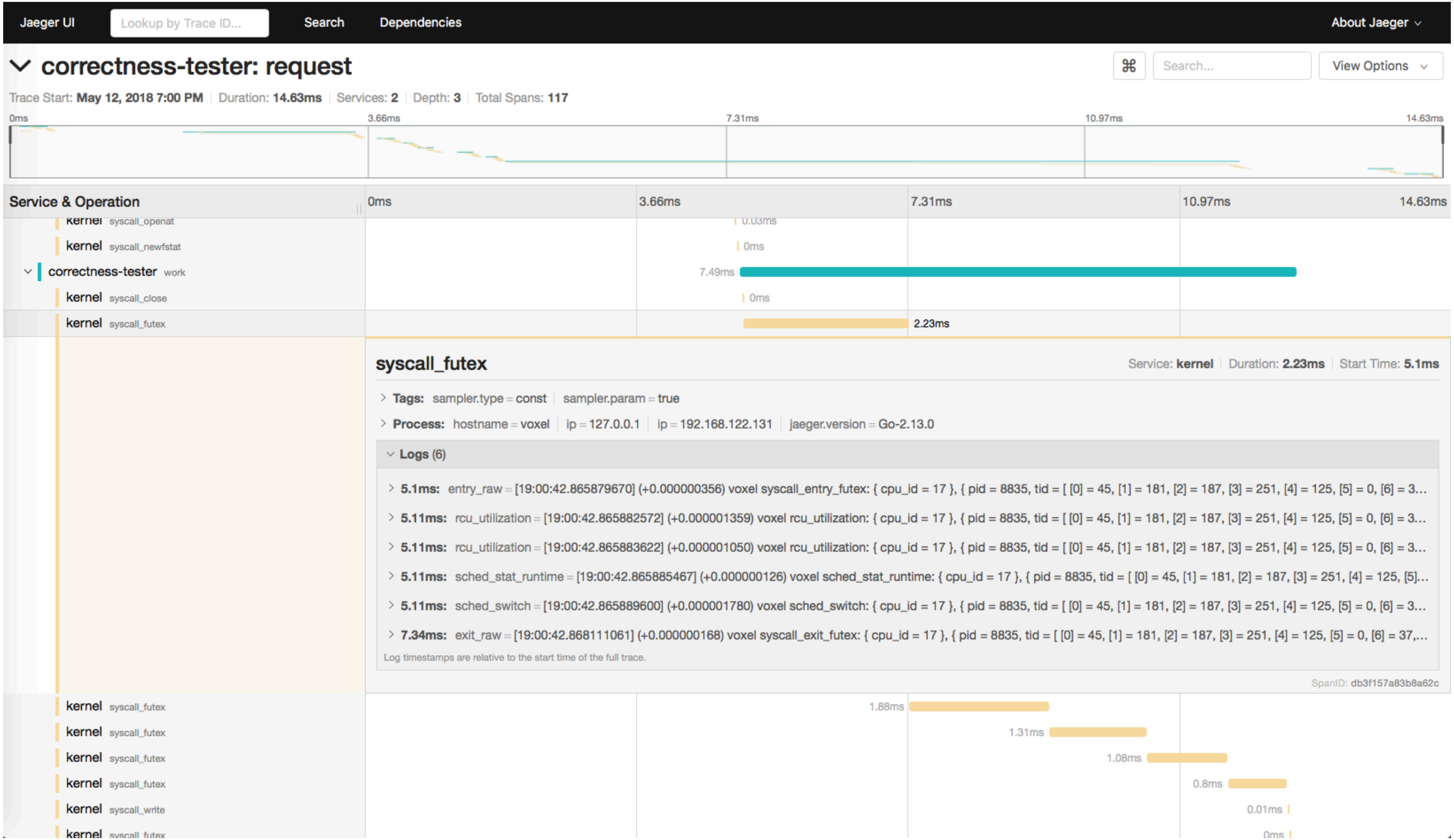
- Raja Sambasivan - Mentor

- MIT PRIMES

Backup Slides

Syscalls made by correctness tester

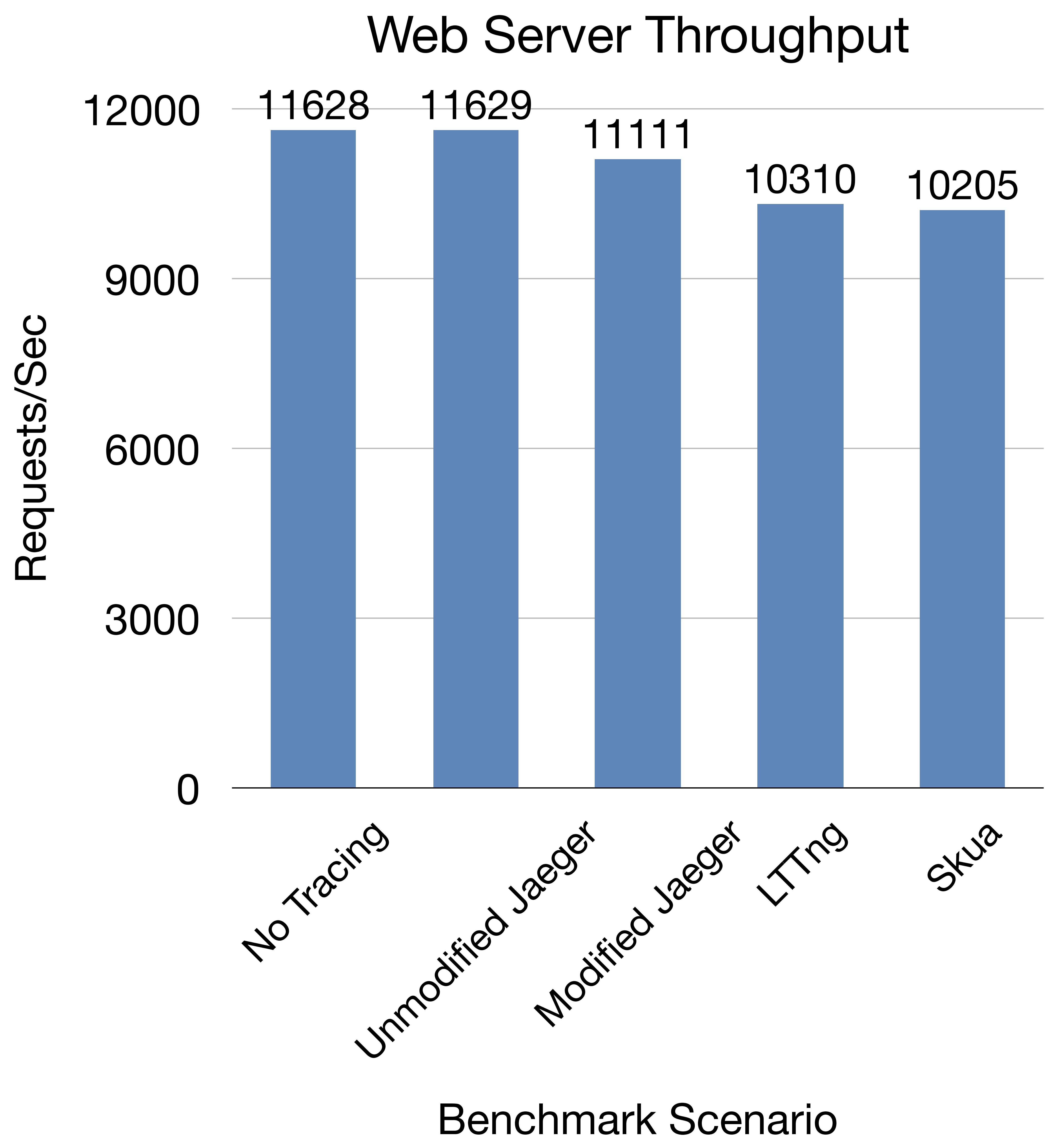
- `getpid`
- `getppid`
- `gettid`
- `gettimeofday`
- `nanosleep`
- `open`
- `close`
- `write`
- `fstat`
- `futex`

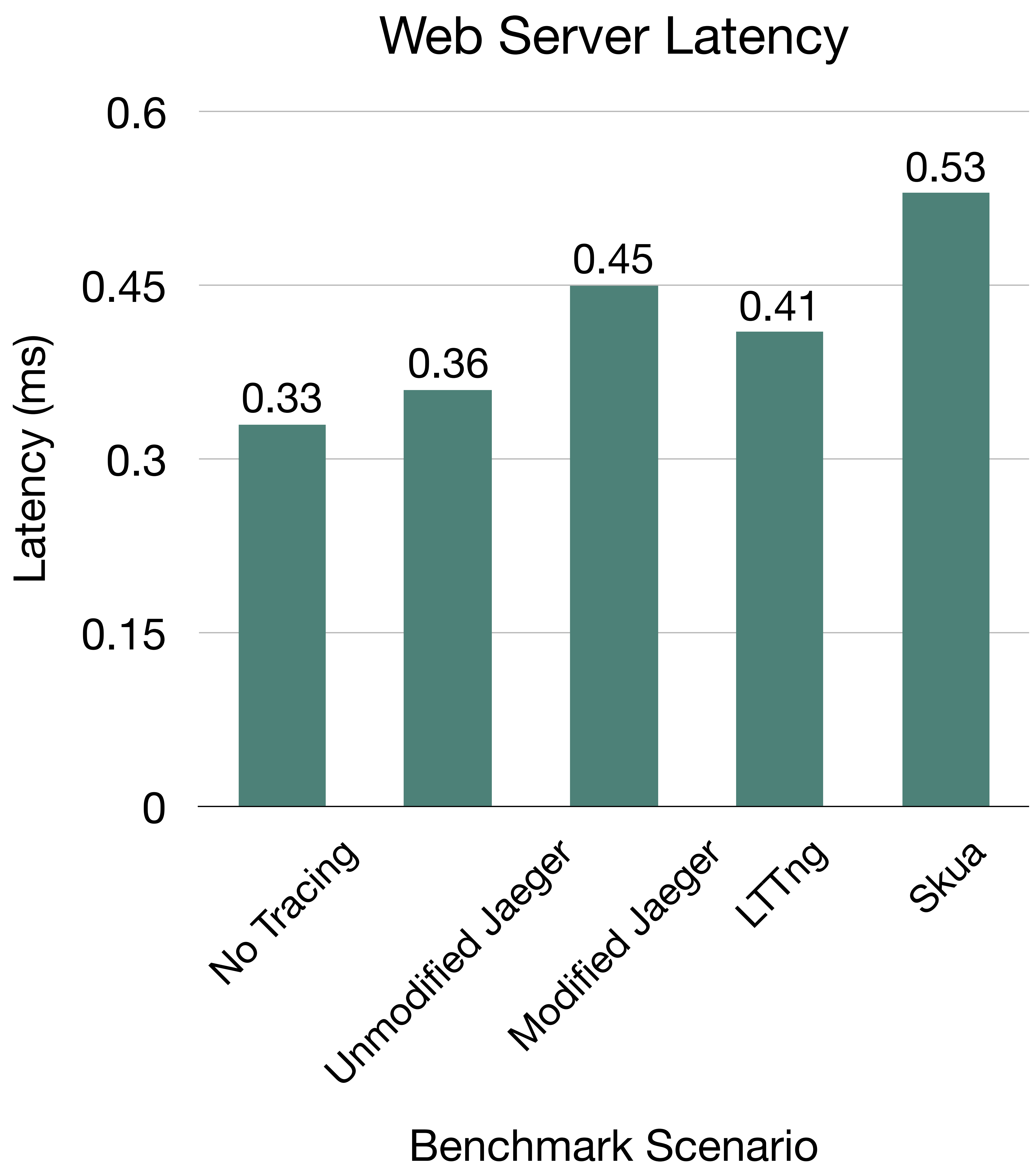

Jaeger UI
Lookup by Trace ID...
Search
Dependencies
About Jaeger
correctness-tester: request
Search...
View Options
Trace Start: May 12, 2018 7:00 PM | Duration: 14.63ms | Services: 2 | Depth: 3 | Total Spans: 117
0ms
3.66ms
7.31ms
10.97ms
14.63ms
Service & Operation
kernel syscall_openat
0.03ms
kernel syscall_newfstat
0ms
correctness-tester work
7.49ms
kernel syscall_close
0ms
kernel syscall_futex
2.23ms
syscall_futex
Service: kernel | Duration: 2.23ms | Start Time: 5.1ms
Tags: sampler.type = const | sampler.param = true
Process: hostname = voxel | ip = 127.0.0.1 | ip = 192.168.122.131 | jaeger.version = Go-2.13.0
Logs (6)
5.1ms: entry_raw = [19:00:42.865879670] (+0.000000356) voxel syscall_entry_futex: { cpu_id = 17 }, { pid = 8835, tid = [[0] = 45, [1] = 181, [2] = 187, [3] = 251, [4] = 125, [5] = 0, [6] = 3...
5.11ms: rcu_utilization = [19:00:42.865882572] (+0.000001359) voxel rcu_utilization: { cpu_id = 17 }, { pid = 8835, tid = [[0] = 45, [1] = 181, [2] = 187, [3] = 251, [4] = 125, [5] = 0, [6] = 3...
5.11ms: rcu_utilization = [19:00:42.865883622] (+0.000001050) voxel rcu_utilization: { cpu_id = 17 }, { pid = 8835, tid = [[0] = 45, [1] = 181, [2] = 187, [3] = 251, [4] = 125, [5] = 0, [6] = 3...
5.11ms: sched_stat_runtime = [19:00:42.865885467] (+0.000000126) voxel sched_stat_runtime: { cpu_id = 17 }, { pid = 8835, tid = [[0] = 45, [1] = 181, [2] = 187, [3] = 251, [4] = 125, [5]...
5.11ms: sched_switch = [19:00:42.865889600] (+0.000001780) voxel sched_switch: { cpu_id = 17 }, { pid = 8835, tid = [[0] = 45, [1] = 181, [2] = 187, [3] = 251, [4] = 125, [5] = 0, [6] = 3...
7.34ms: exit_raw = [19:00:42.868111061] (+0.000000168) voxel syscall_exit_futex: { cpu_id = 17 }, { pid = 8835, tid = [[0] = 45, [1] = 181, [2] = 187, [3] = 251, [4] = 125, [5] = 0, [6] = 37,...
Log timestamps are relative to the start time of the full trace.
SpanID: db3f157a83b8a62c
kernel syscall_futex
1.88ms
kernel syscall_futex
1.31ms
kernel syscall_futex
1.08ms
kernel syscall_futex
0.8ms
kernel syscall_write
0.01ms
kernel syscall_futex
0ms

Performance Benchmark Details

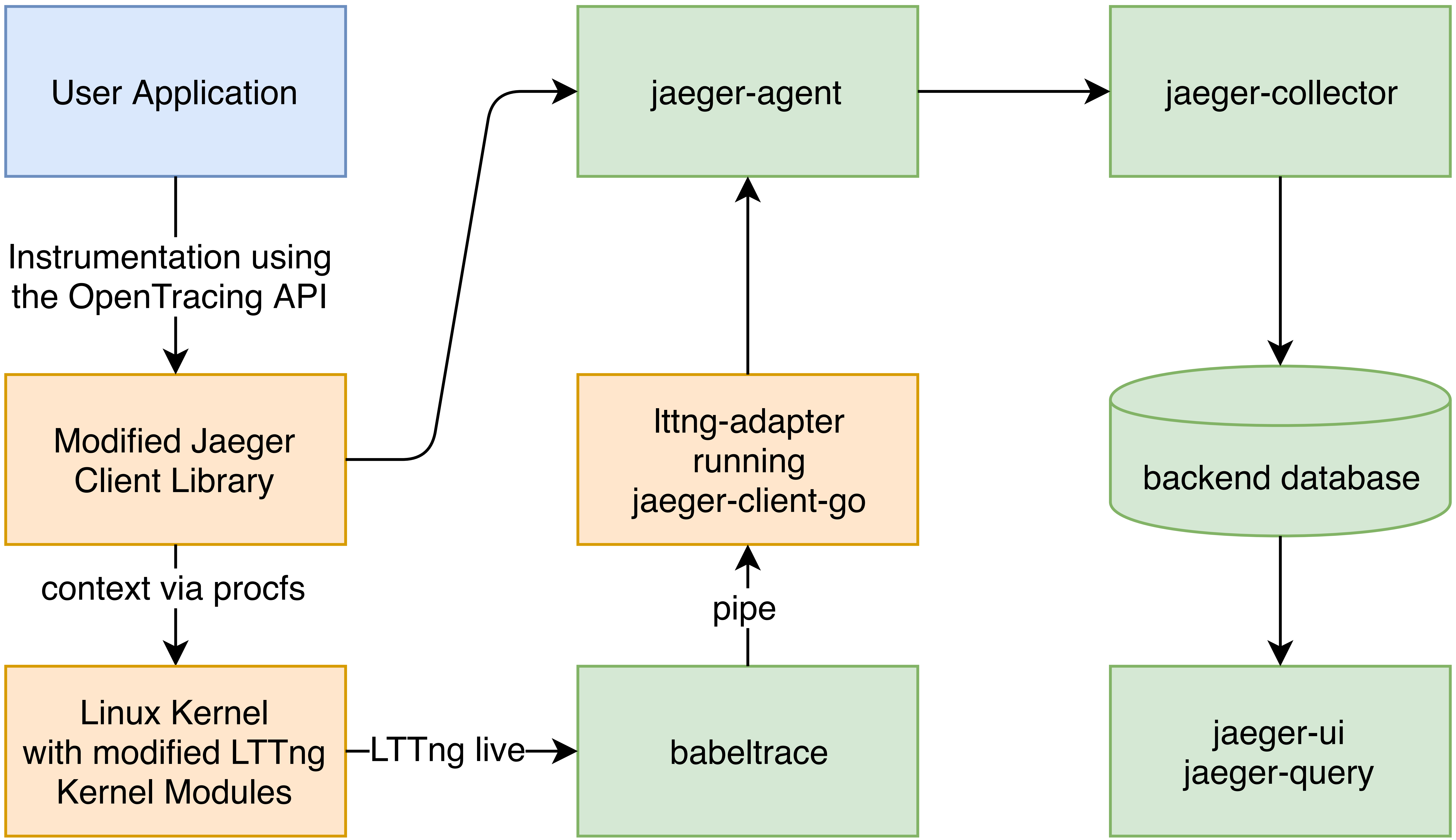
- Tests run on KVM virtual machine assigned 24 vCPUs (2 × Intel Xeon X5670), 16 GB RAM, with latest Arch Linux updates

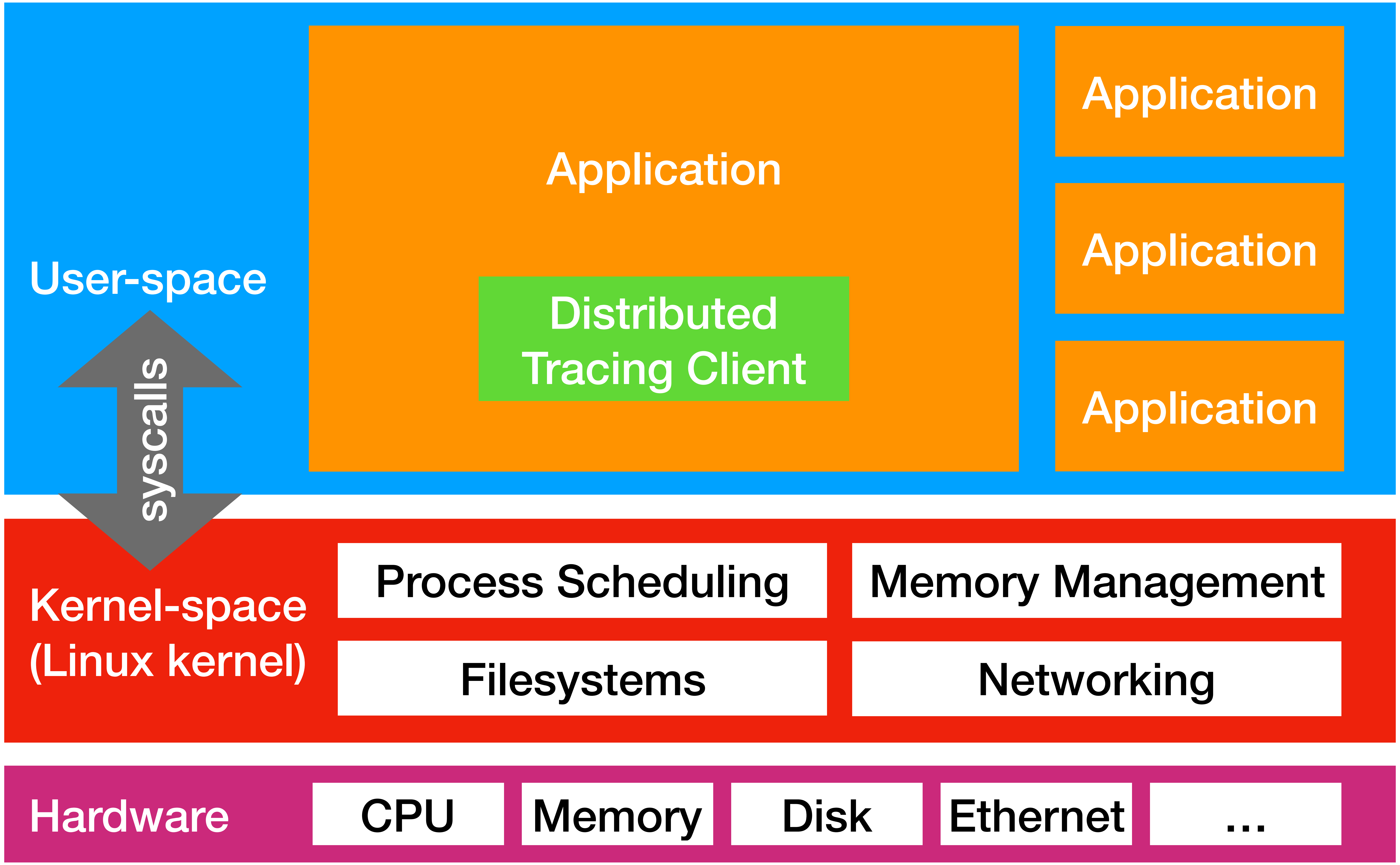
Benchmark Scenario	Program Instrumentation	Kernel Tracing via LTTng
No Tracing	N/A	No
Unmodified Jaeger	Original Jaeger Client	No
Modified Jaeger	Modified Jaeger Client	No
LTTng	N/A	Yes, but output ignored
Skua	Modified Jaeger Client	Yes, output sent to adapter


Web Server Throughput
Requests/Sec
12000
9000
6000
3000
0
11628
11629
11111
10310
10205
No Tracing
Unmodified Jaeger
Modified Jaeger
LTTng
Skua
Benchmark Scenario


Web Server Latency
Latency (ms)
0.6
0.45
0.3
0.15
0
0.33
0.36
0.45
0.41
0.53
No Tracing
Unmodified Jaeger
Modified Jaeger
LTTng
Skua
Benchmark Scenario

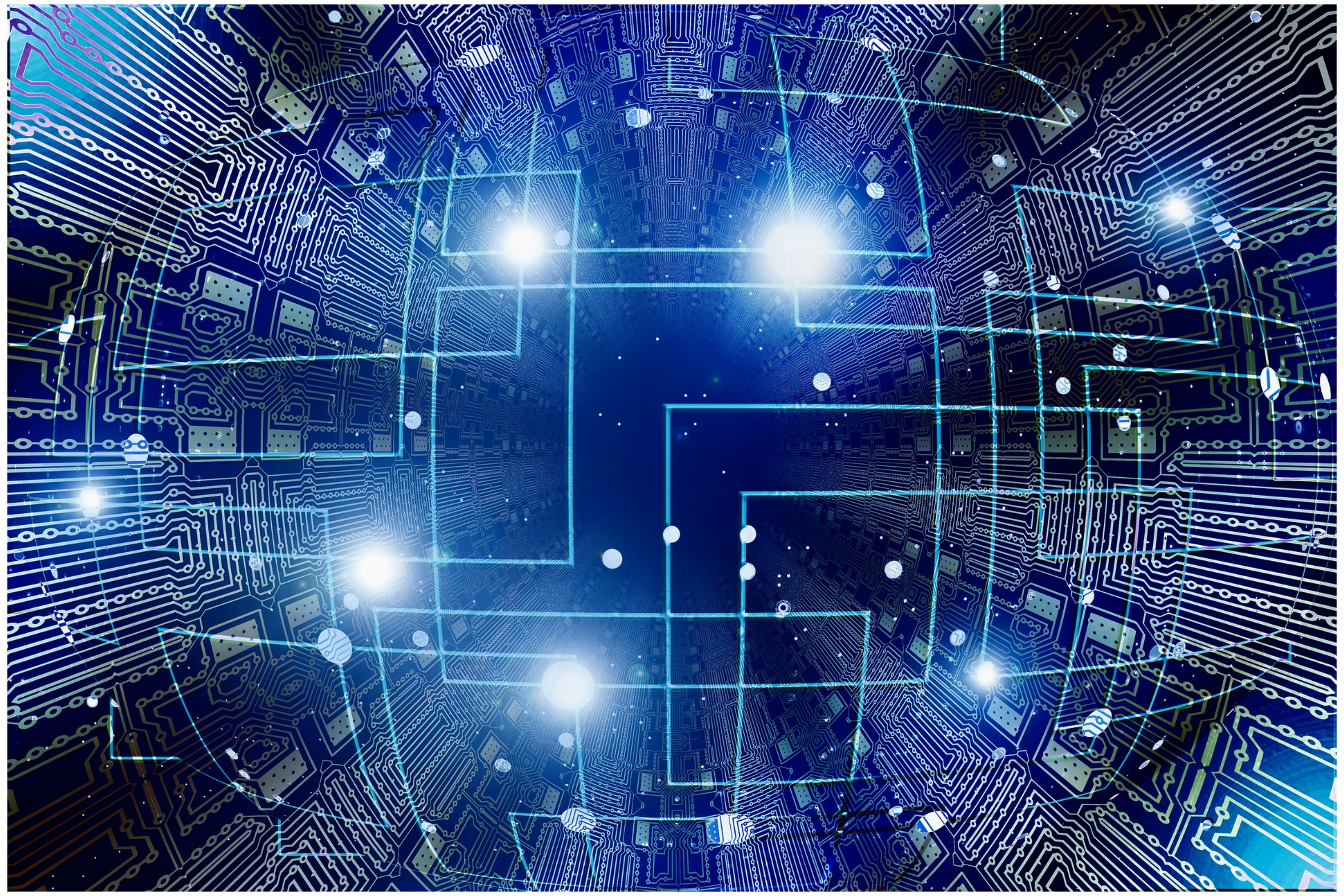
Skua Implementation Details



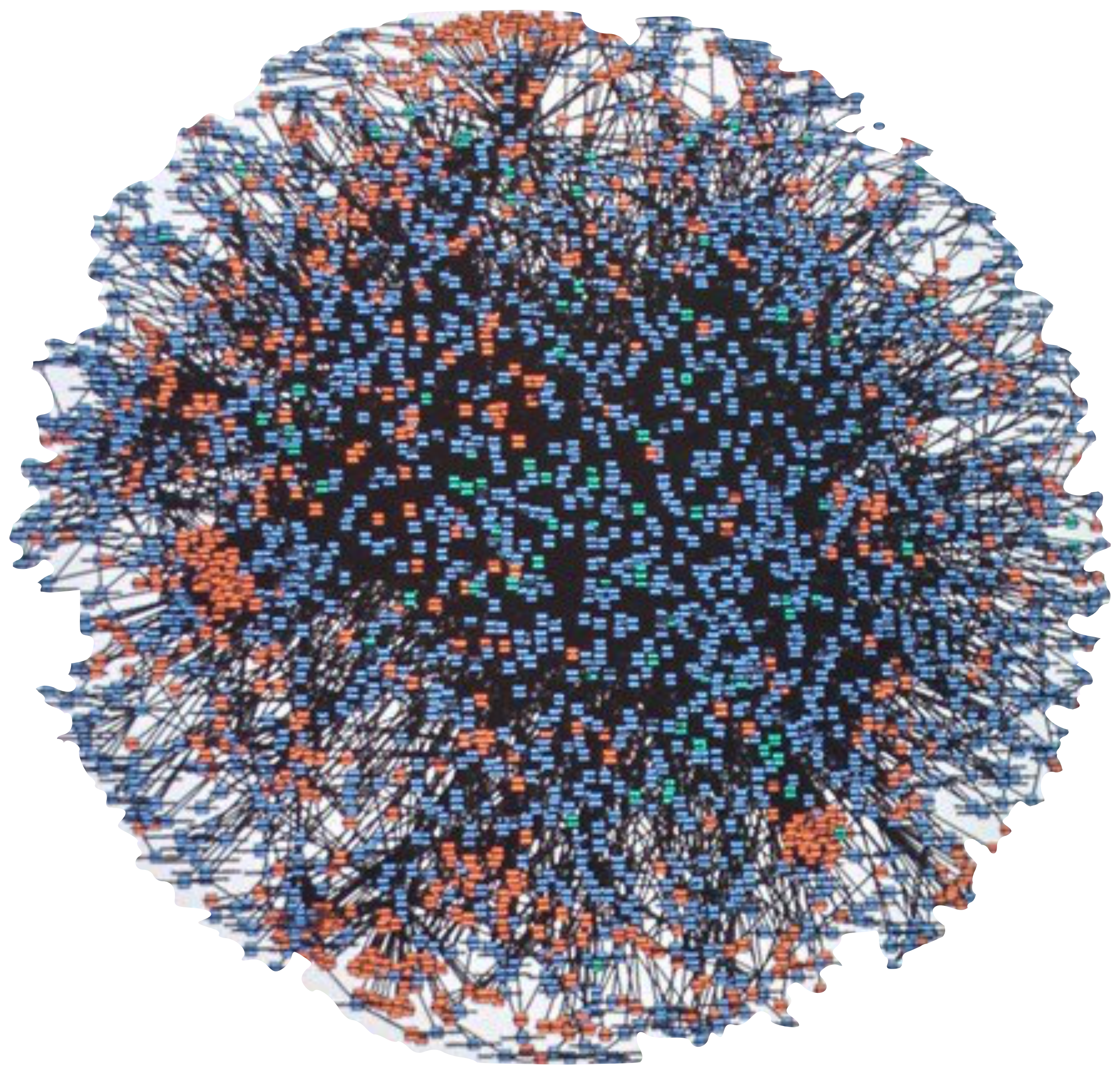

Application
Application
Application
Application
Distributed Tracing Client
User-space
syscalls
Kernel-space (Linux kernel)
Process Scheduling
Memory Management
Filesystems
Networking
Hardware
CPU
Memory
Disk
Ethernet
…

Existing Tracing Frameworks

- Dapper (2010) — Google
- Zipkin (2012) —Twitter
- Canopy (2017) — Facebook
- Jaeger (2017) — open sourced by Uber

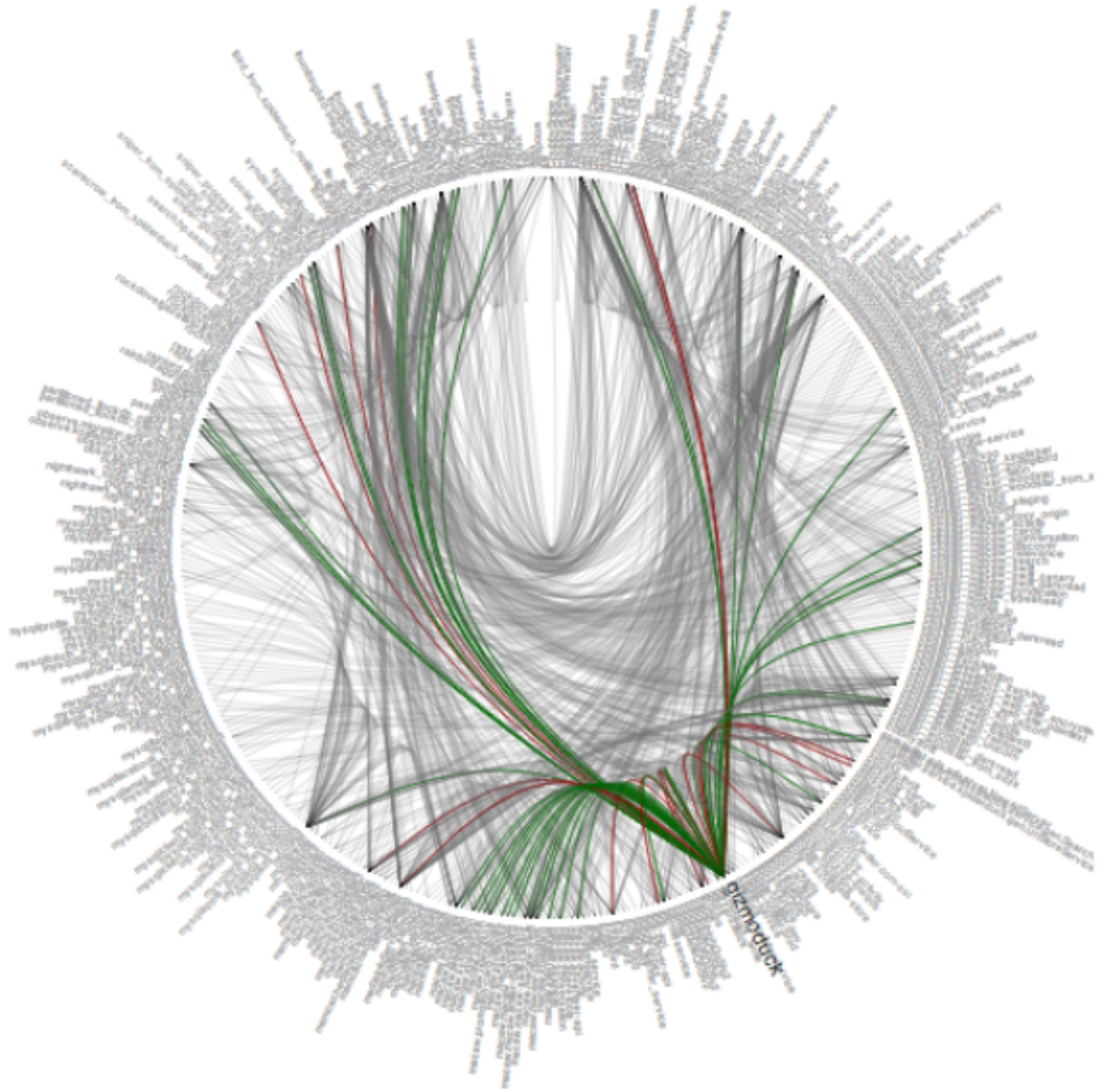


Distributed Systems



AWS (2008)

Netflix (2012)



Twitter (2013)

AWS: https://twitter.com/werner/status/741673514567143424
Netflix: https://www.slideshare.net/BruceWong3/the-case-for-chaos
Twitter: https://blog.twitter.com/engineering/en_us/a/2013/observability-at-twitter.html