Request Reconstruction in MirageOS Unikernels

Al Amjad Tawfiq Isstaif (aati2)
University of Cambridge
21 Jun 2019

Motivation

Exploring the potential of combining unikernels with distributed tracing to address the autoscaling problem in microservice applications

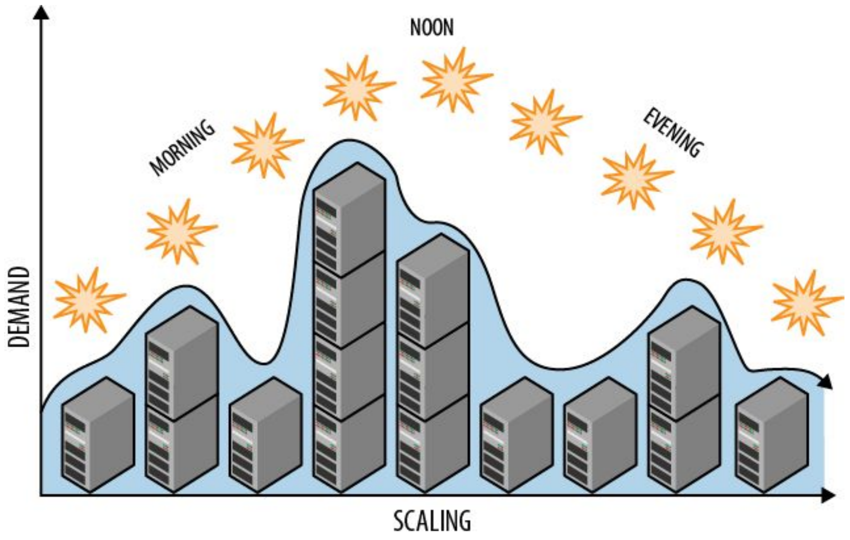
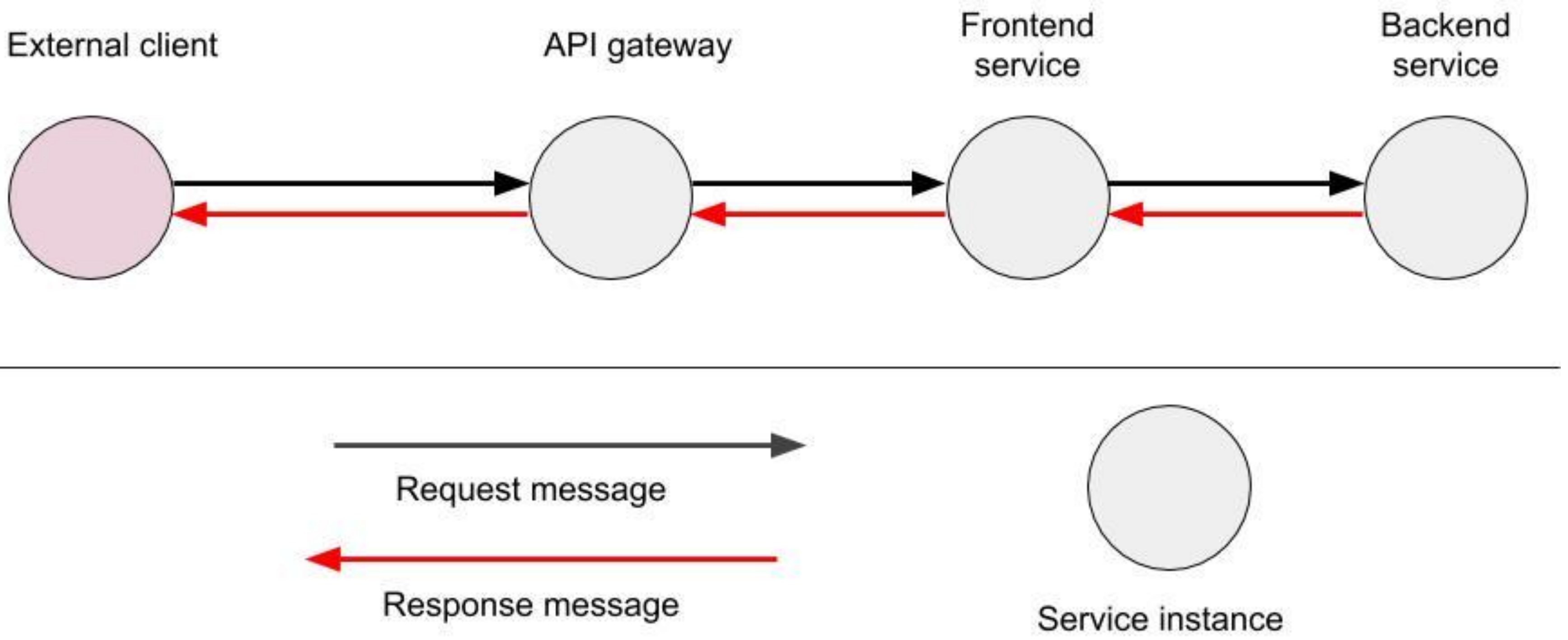
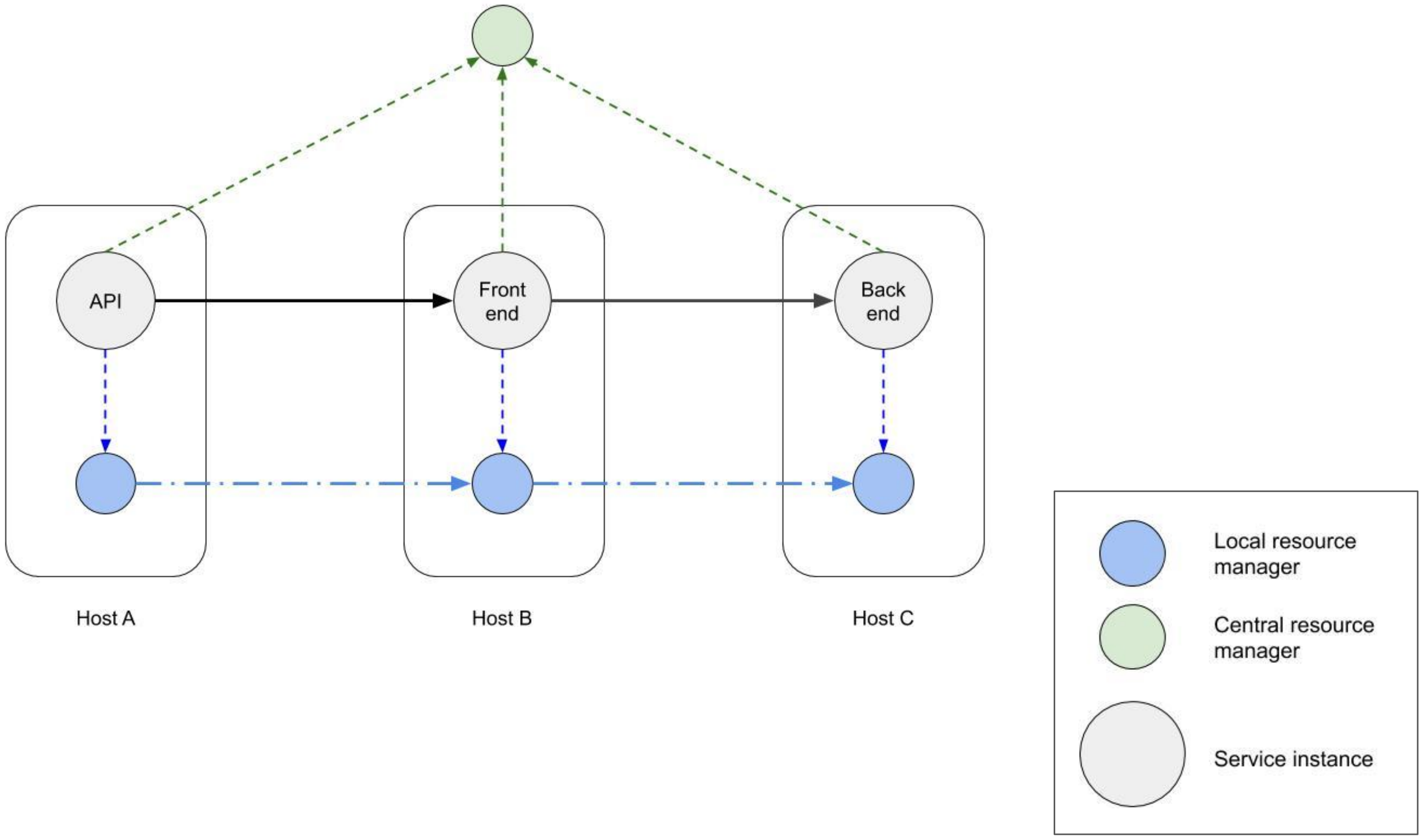


Image sources: O’Reilly Media, Netflix

Running example



Trace collection and resource management



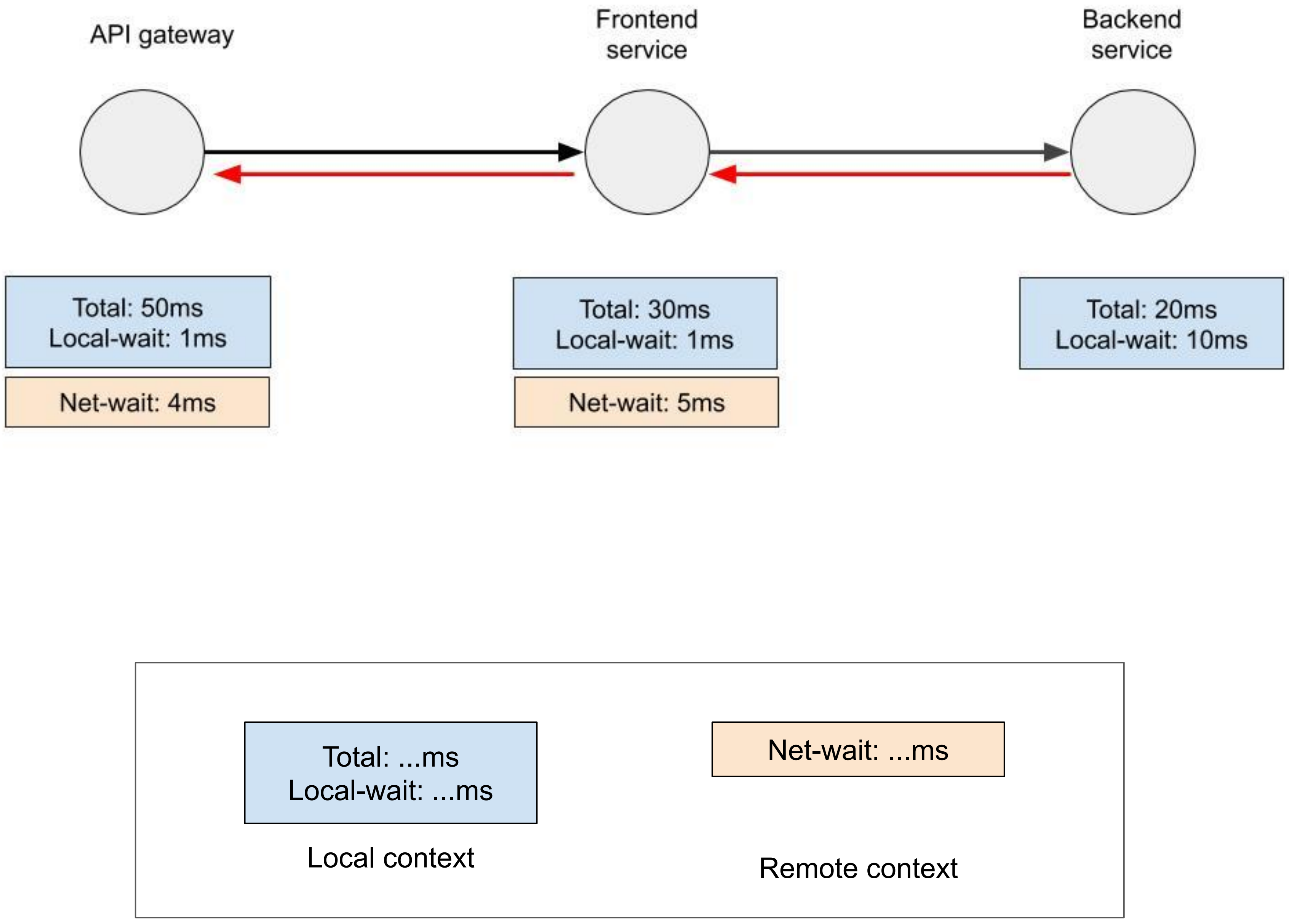
Key ideas

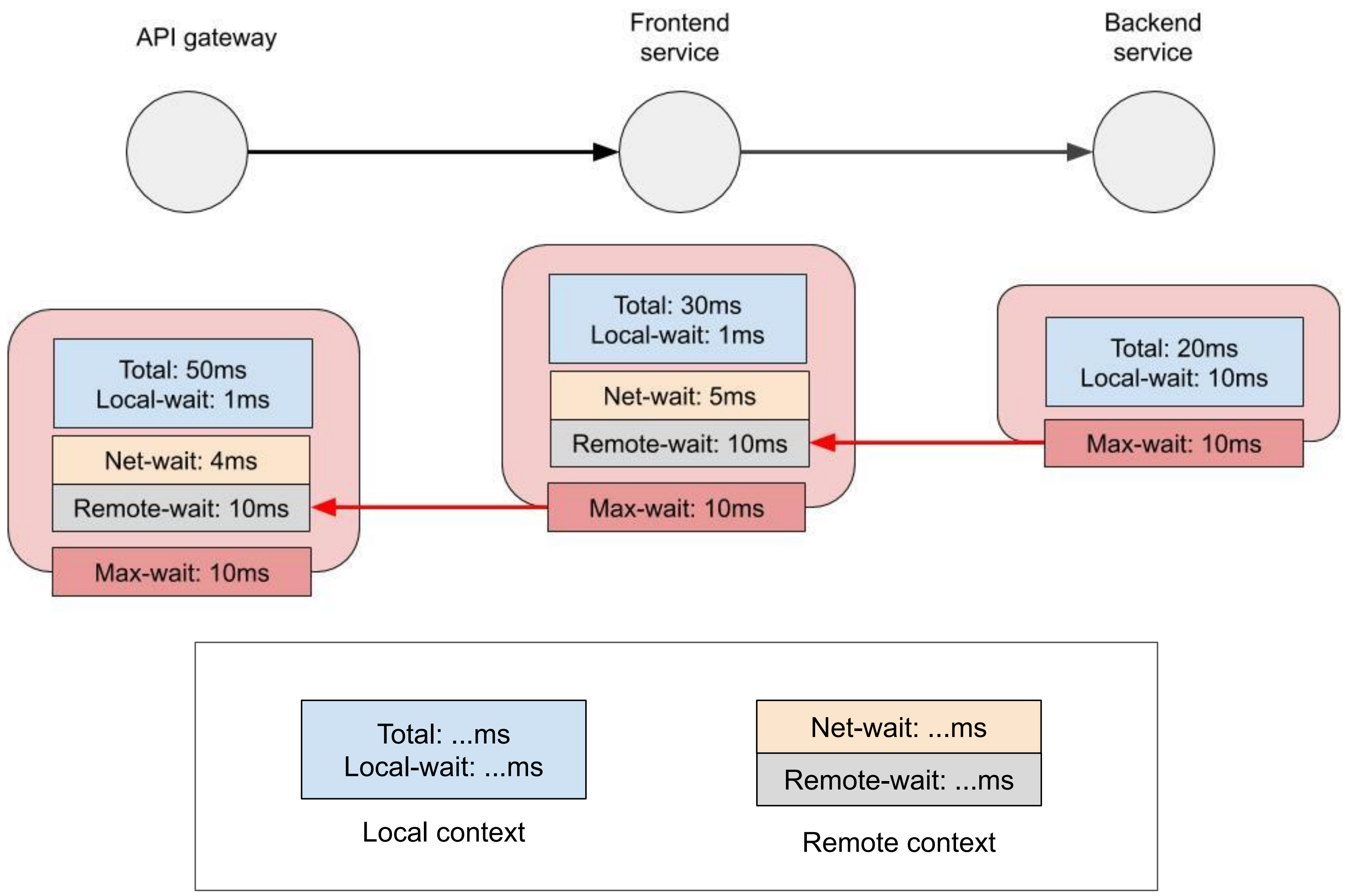
Key ideas in the proposed tracing model:

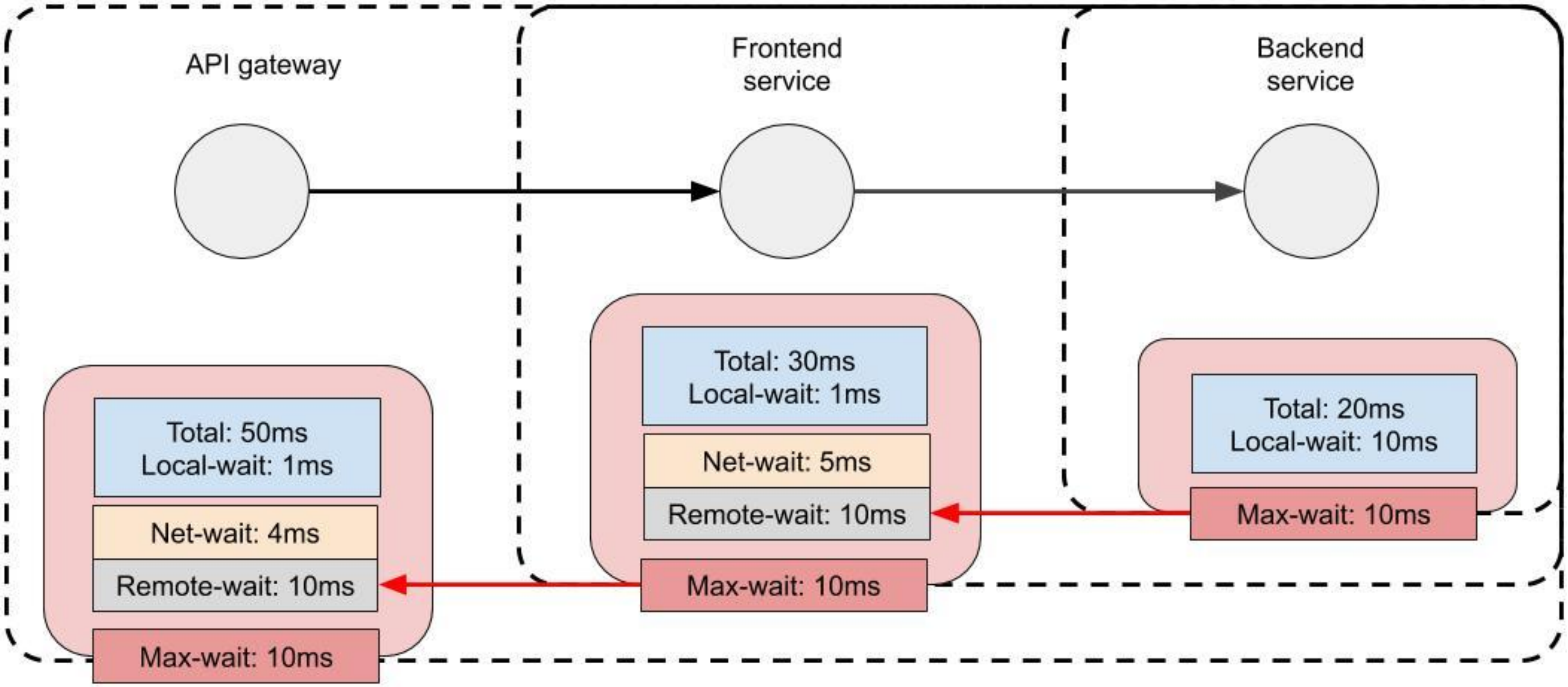
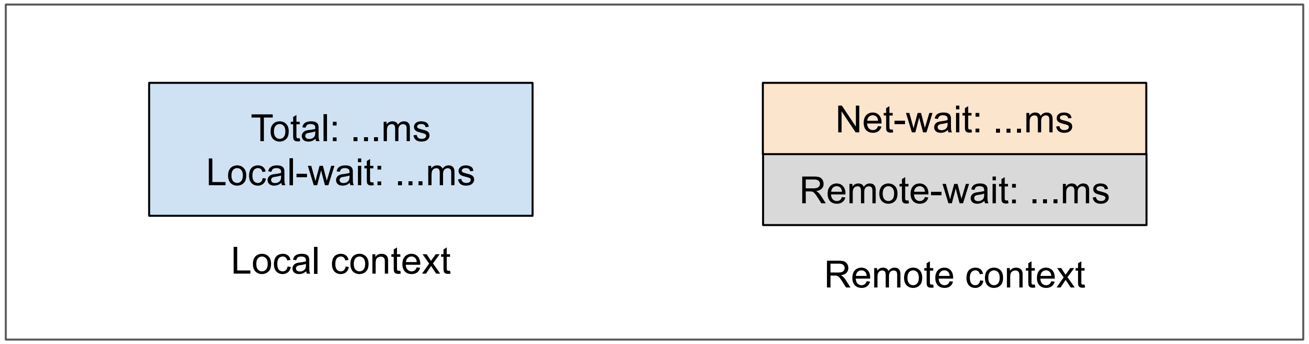
- Measuring resource waiting times (CPU and Network)
- “Flexible” tracing analysis (application-level & service-level trace analysis)

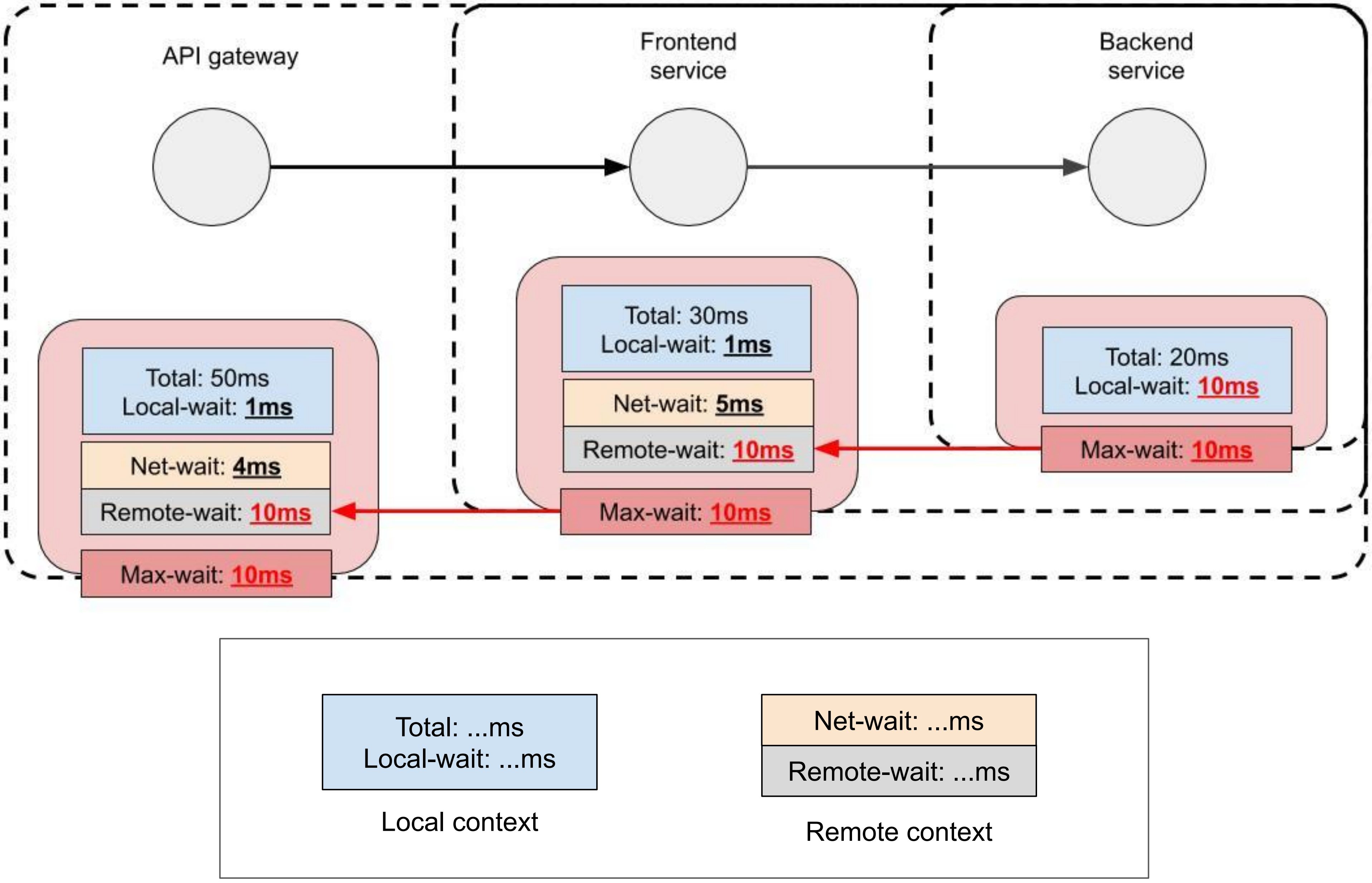
Generic execution wrappers to instrument Mirage HTTP library (Cohttp)

- Protocol agnostic: can be used to automatically instrument any protocol libraries based on inter-service communication
- Mirage libraries based on Lwt lightweight threads or *promises (futures)*

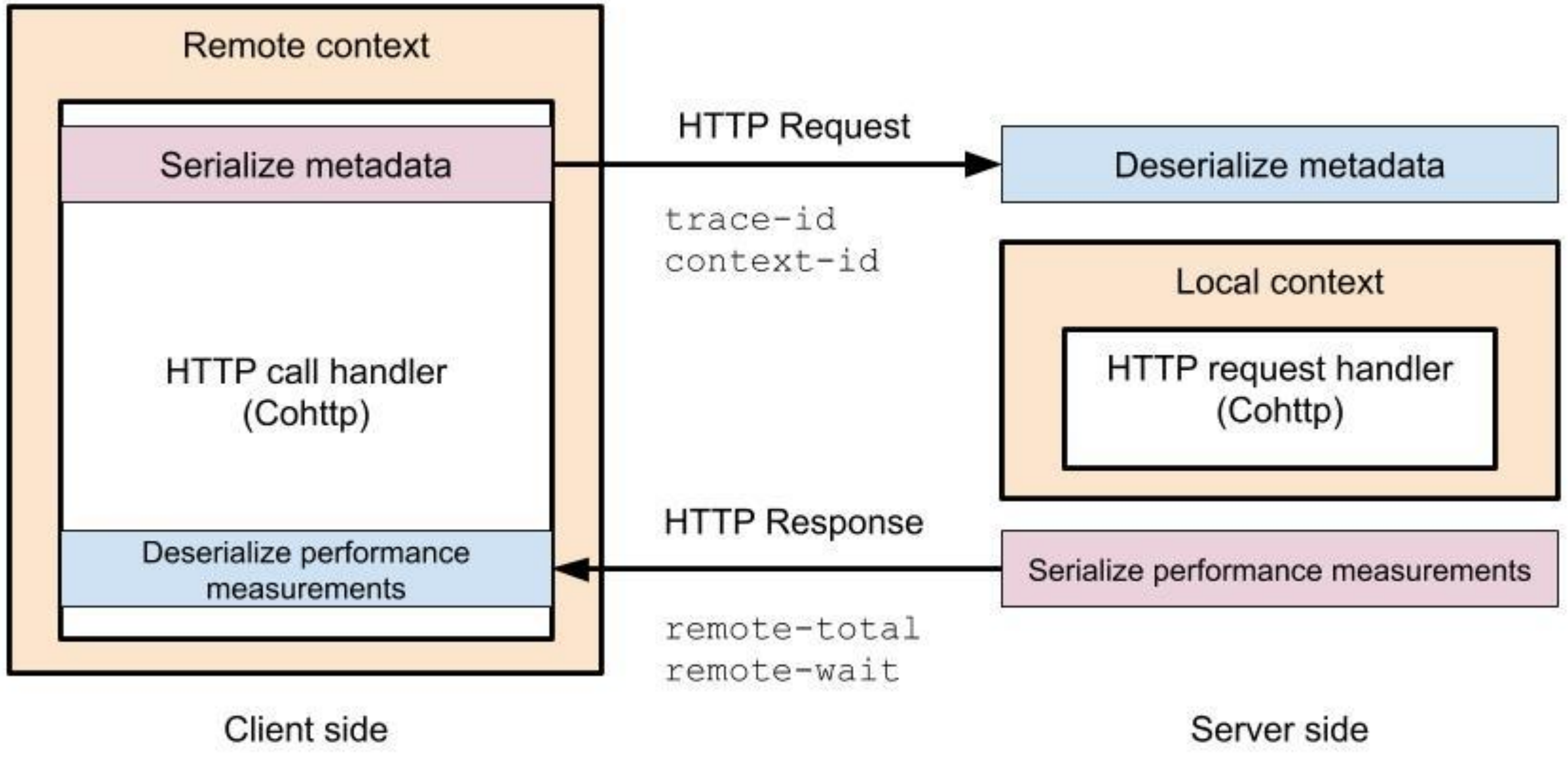

API gateway
Frontend service
Backend service
Total: 50ms
Local-wait: 1ms
Net-wait: 4ms
Total: 30ms
Local-wait: 1ms
Net-wait: 5ms
Total: 20ms
Local-wait: 10ms
Total: ...ms
Local-wait: ...ms
Local context
Net-wait: ...ms
Remote context


API gateway
Frontend service
Backend service
Total: 50ms
Local-wait: 1ms
Net-wait: 4ms
Remote-wait: 10ms
Max-wait: 10ms
Total: 30ms
Local-wait: 1ms
Net-wait: 5ms
Remote-wait: 10ms
Max-wait: 10ms
Total: 20ms
Local-wait: 10ms
Max-wait: 10ms
Total: ...ms
Local-wait: ...ms
Local context
Net-wait: ...ms
Remote-wait: ...ms
Remote context


API gateway
Frontend service
Backend service
Total: 50ms
Local-wait: 1ms
Net-wait: 4ms
Remote-wait: 10ms
Max-wait: 10ms
Total: 30ms
Local-wait: 1ms
Net-wait: 5ms
Remote-wait: 10ms
Max-wait: 10ms
Total: 20ms
Local-wait: 10ms
Max-wait: 10ms

Total: ...ms
Local-wait: ...ms
Local context
Net-wait: ...ms
Remote-wait: ...ms
Remote context


API gateway
Frontend service
Backend service
Total: 50ms
Local-wait: 1ms
Net-wait: 4ms
Remote-wait: 10ms
Max-wait: 10ms
Total: 30ms
Local-wait: 1ms
Net-wait: 5ms
Remote-wait: 10ms
Max-wait: 10ms
Total: 20ms
Local-wait: 10ms
Max-wait: 10ms
Total: ...ms
Local-wait: ...ms
Local context
Net-wait: ...ms
Remote-wait: ...ms
Remote context

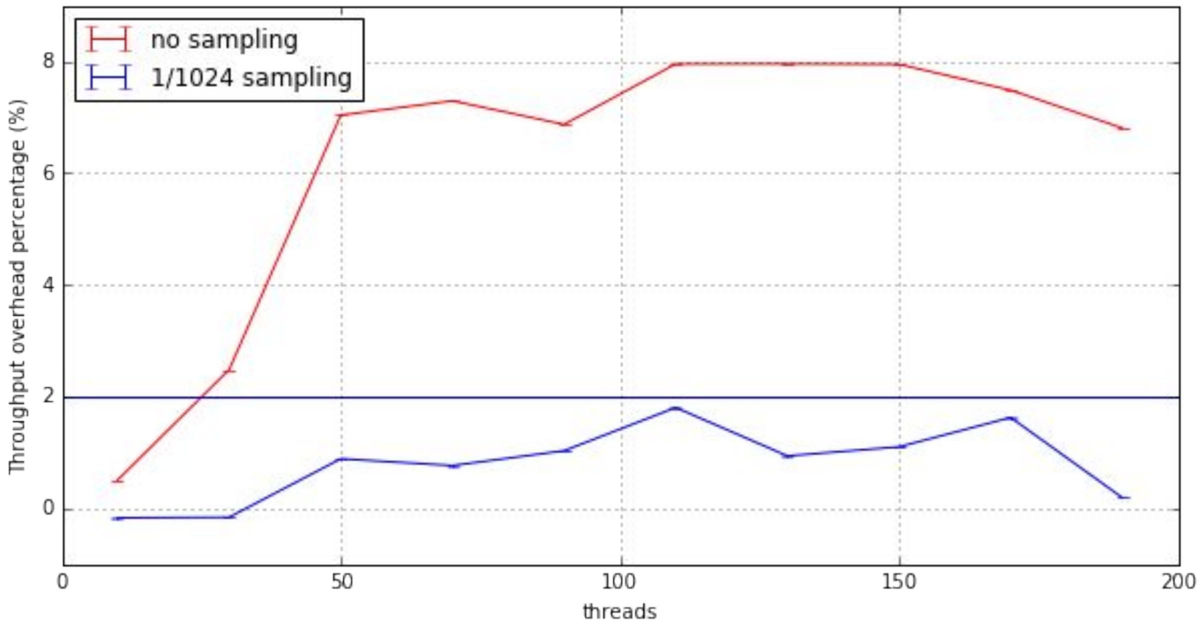
Execution wrappers and Cohttp library instrumentation



Overhead evaluation

For 100 explicit context switch points (also known as *yield points*)

- 8% overhead with no sampling
- Less than 2% under a sampling rate of 1/1024



Summary and conclusions

A tracing model based on hierarchical aggregation of resource waiting times suitable for further research on microservices autoscaling policies

- The tracing model can be extended for other lightweight virtualization options (e.g. containers)

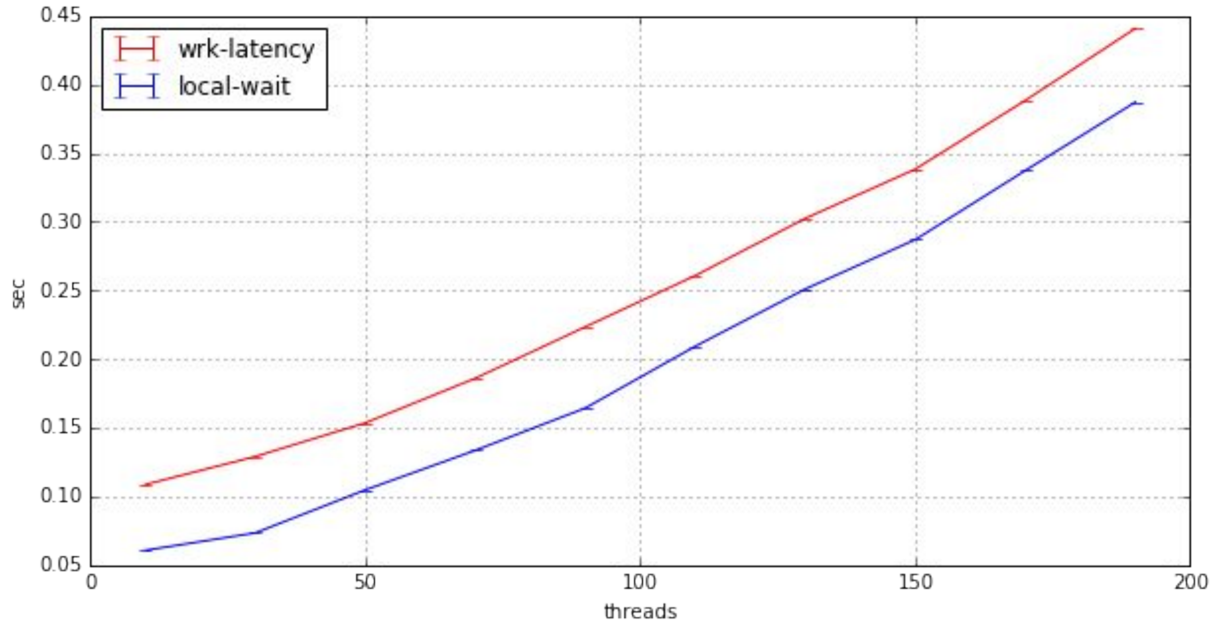
The single-threaded architecture of Mirage unikernels introduce various issues that need to be addressed in future work

- A CPU bottleneck can lead to an indistinguishable artificial network bottleneck
- Instrumenting the TCP stack can be useless, and packet-timestamping is required

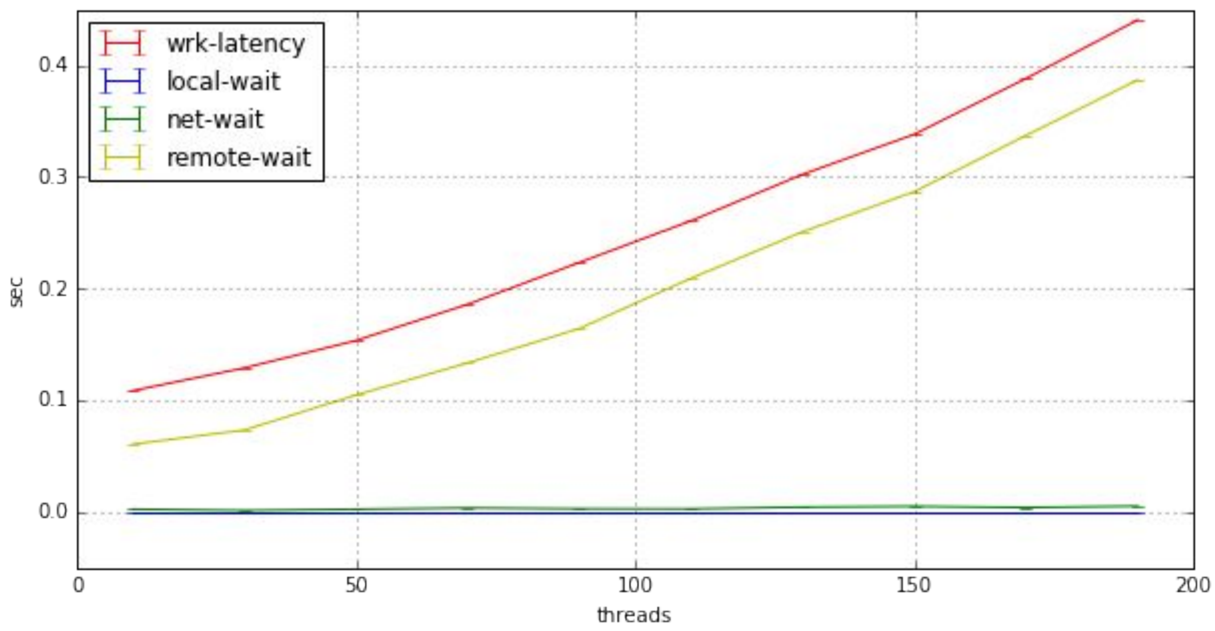
Backup slides

CPU bottleneck

Backend service

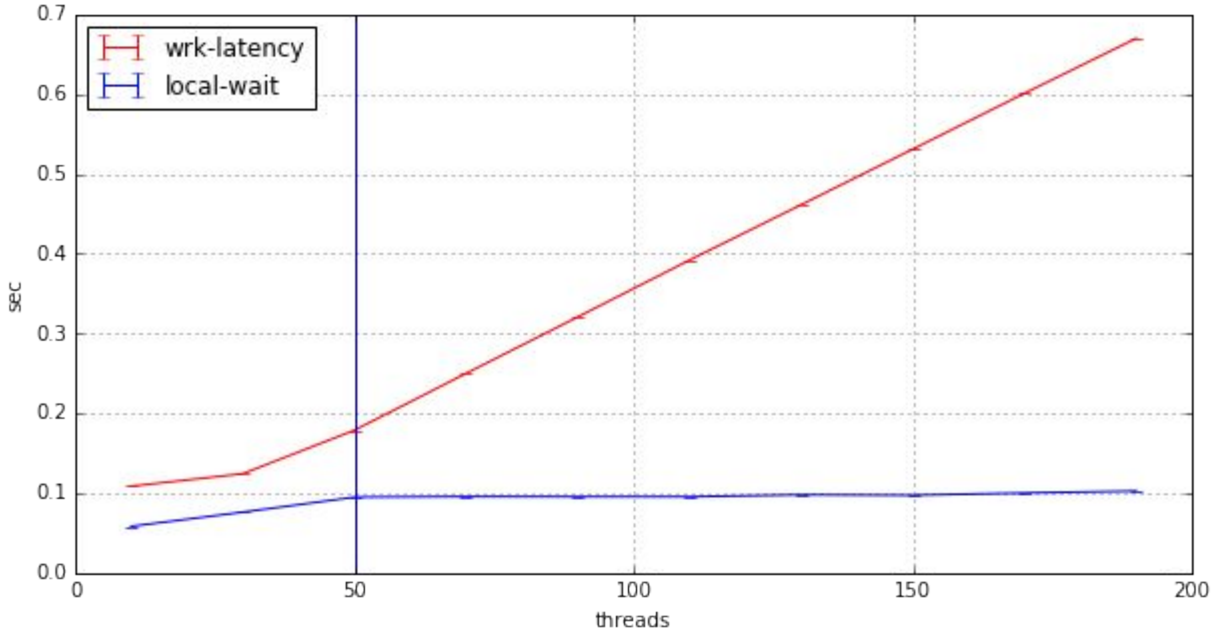


Frontend service

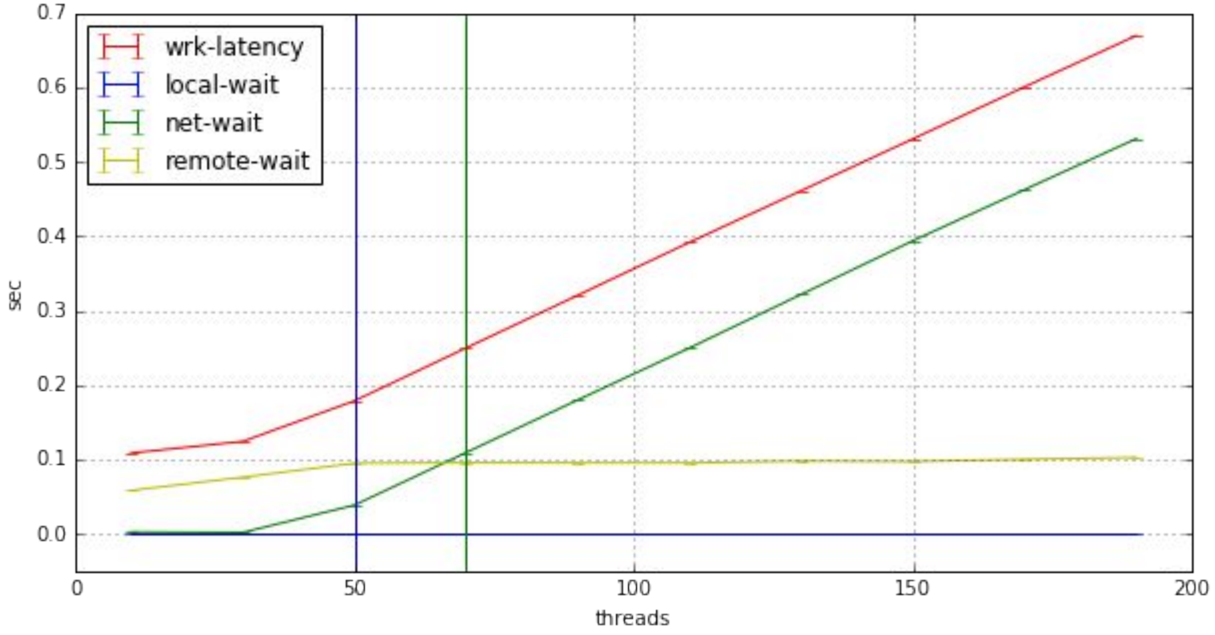


Net bottleneck

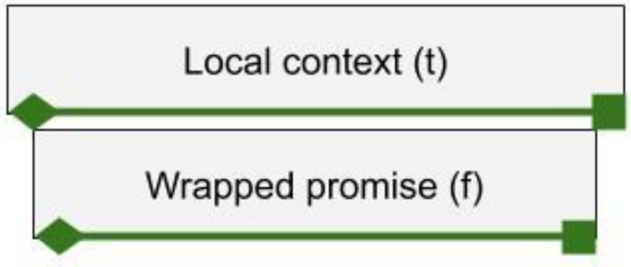
Backend service



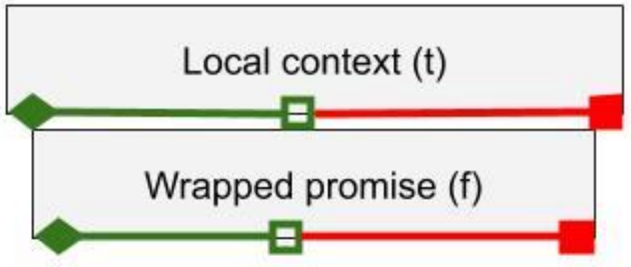
Frontend service



Implementation of local context wrapper

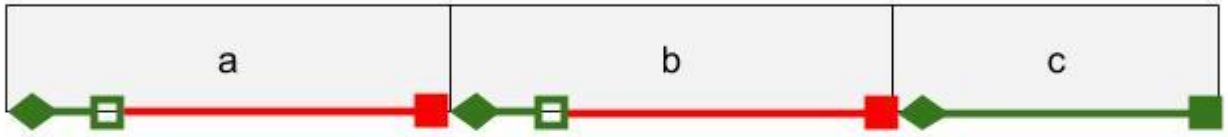


Immediately resolving promise



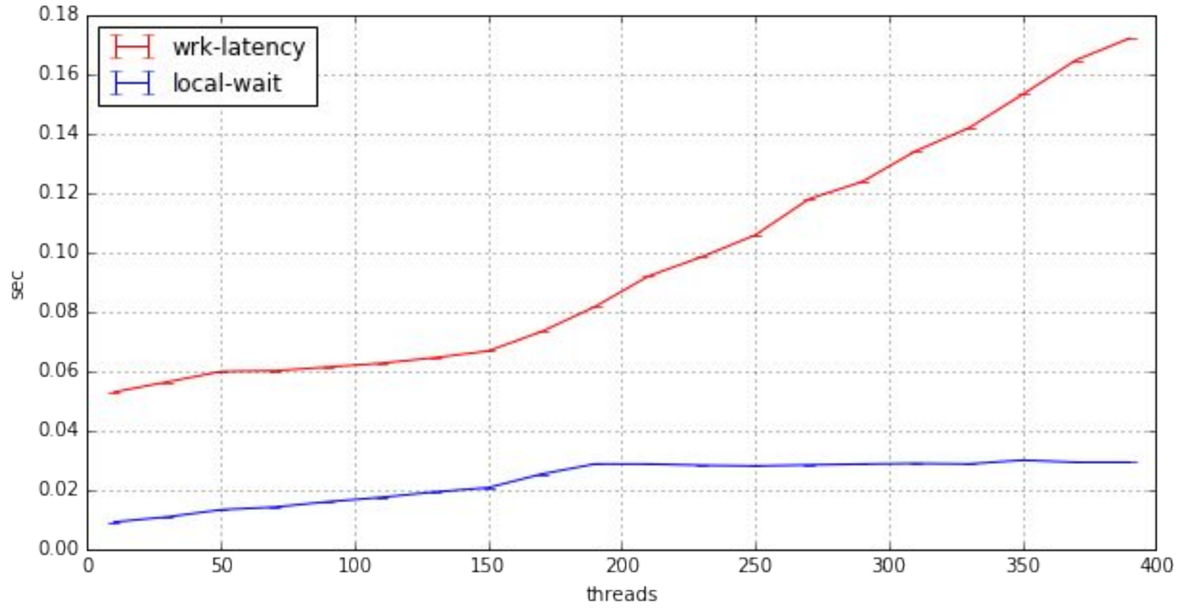
Asynchronous promise

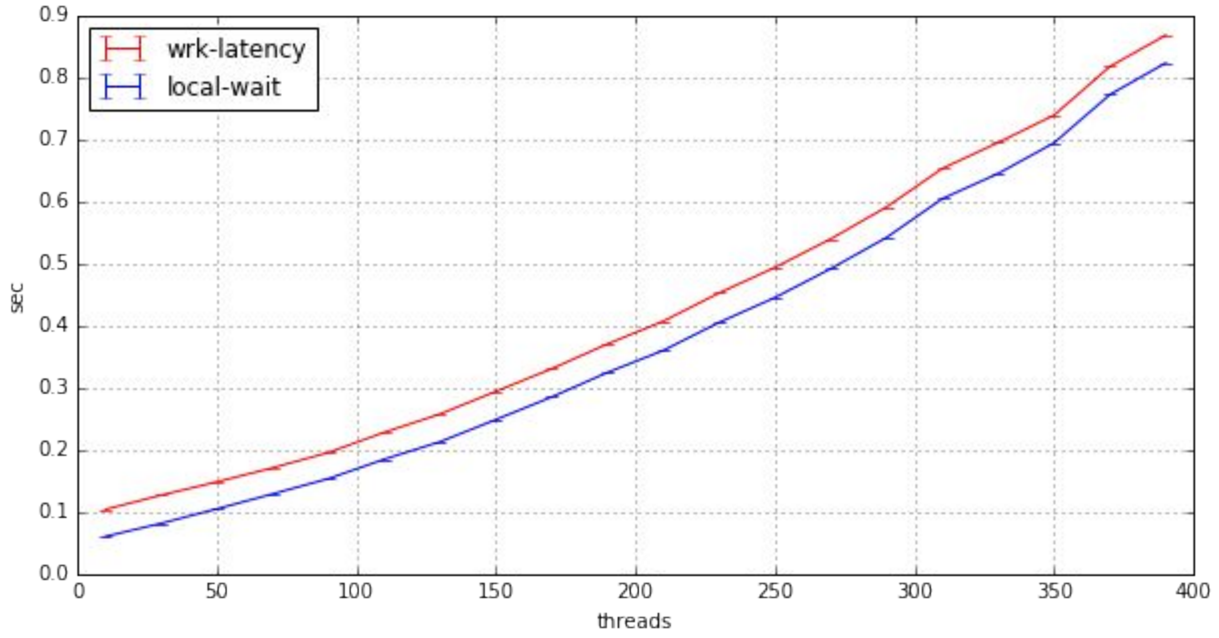
```
let t = Lwt.with_context (fun () -> f)
```



```
let t = Lwt.with_context (fun () -> a >>= b >>= c)
```

Avoiding an artificial network bottleneck (above) by increasing the number of context switches (below)


wrk-latency
local-wait
sec
threads


wrk-latency
local-wait
sec
threads