P99 CONF

Why User-Mode Threads Are Good for Performance

Ron Pressler
Architect, Java Platform Group, Oracle

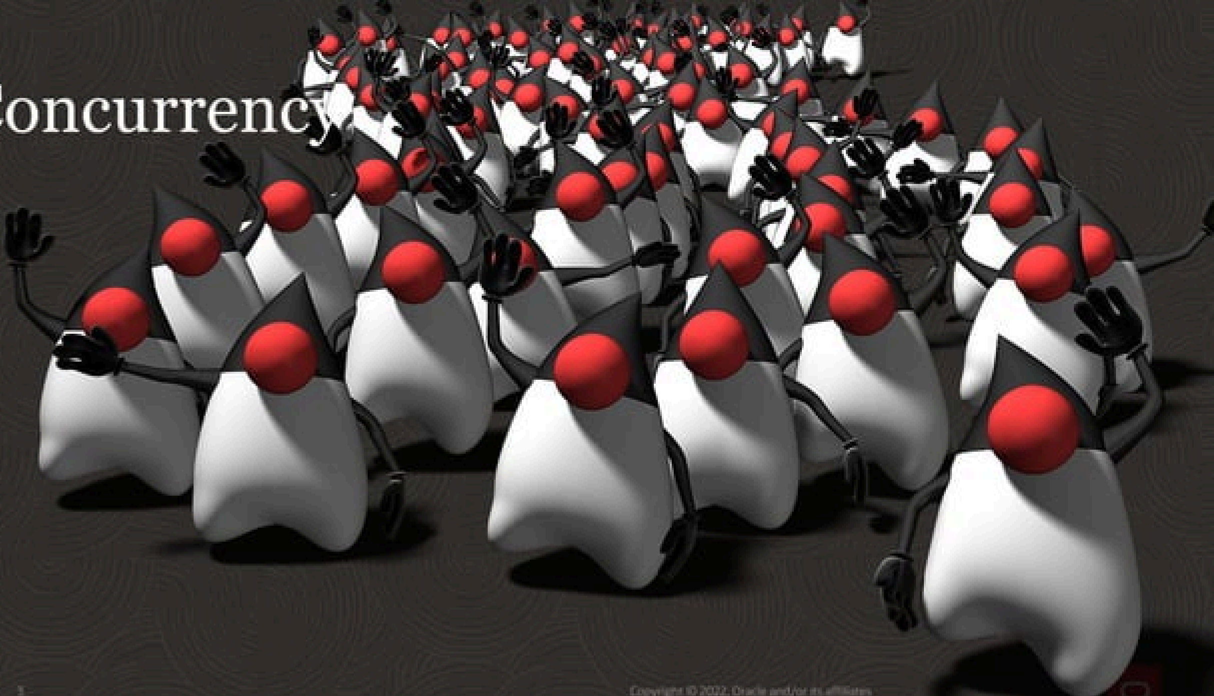
Brought to you by SCYLLA

Why?

- Why do anything?
- ~~Why do *this* rather than *that*?~~

Copyright © 2022, Oracle and/or its affiliates

Concurrency



Copyright © 2022, Oracle and/or its affiliates

Concurrency and Parallelism

- *Parallelism*:
  - Speed up a task by splitting it into *cooperating* subtasks scheduled onto multiple available computing resources.
  - Performance measure: latency (time duration)
- *Concurrency*:
  - Schedule available computing resources to multiple largely independent tasks that *compete* over them.
  - Performance measure: throughput (task/time unit)

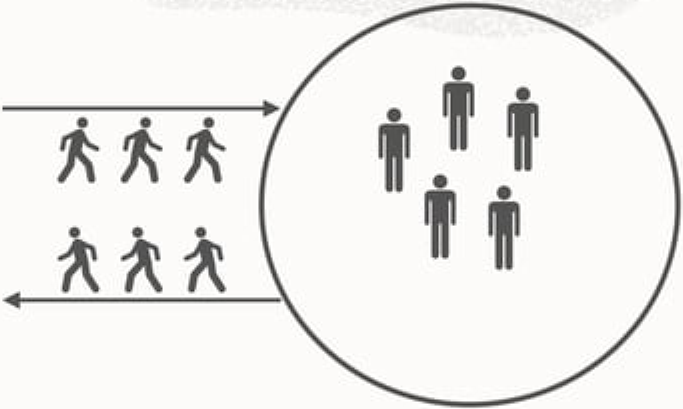
Copyright © 2022, Oracle and/or its affiliates

Little’s Law

Copyright © 2022, Oracle and/or its affiliates

Little's Law

In *any* ***stable*** system



Copyright © 2022, Oracle and/or its affiliates

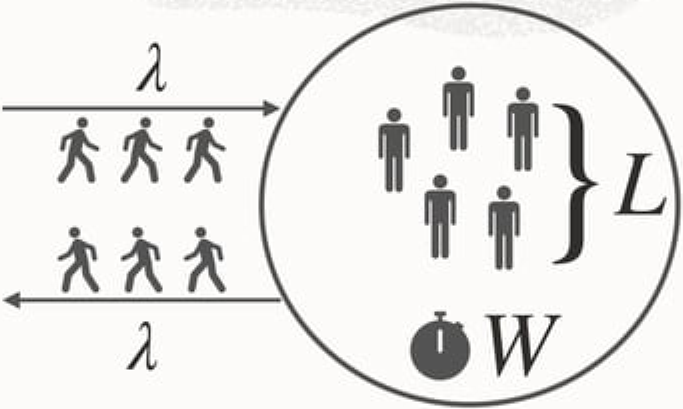
Little’s Law

In *any* ***stable*** system
with *long term averages*:

λ — arrival rate = exit rate
= throughput

W — duration inside

L — no. items inside

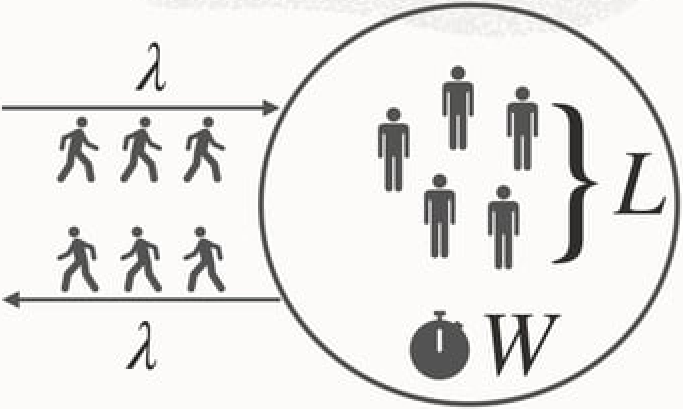


Copyright © 2022, Oracle and/or its affiliates

Little's Law

In *any* ***stable*** system
with *long term averages*:

λ — arrival rate = exit rate
= throughput
W — duration inside
L — no. items inside

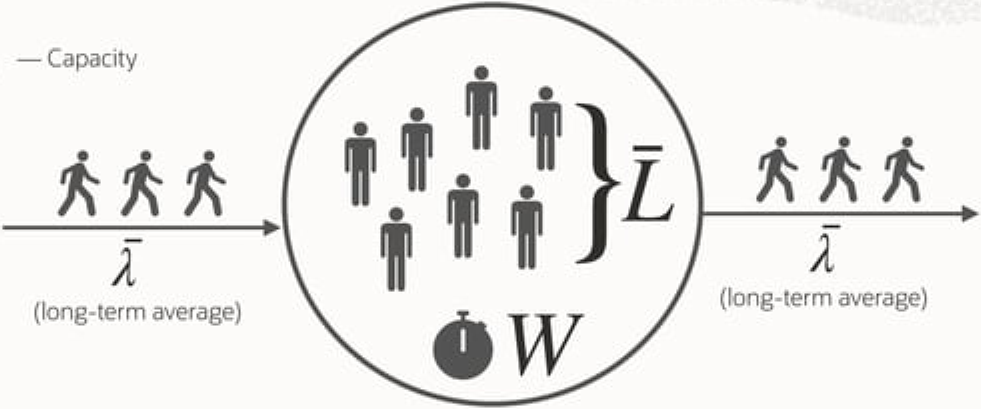


$$L = \lambda W$$

Copyright © 2022, Oracle and/or its affiliates

Little's Law

$\bar{L}$ — Capacity

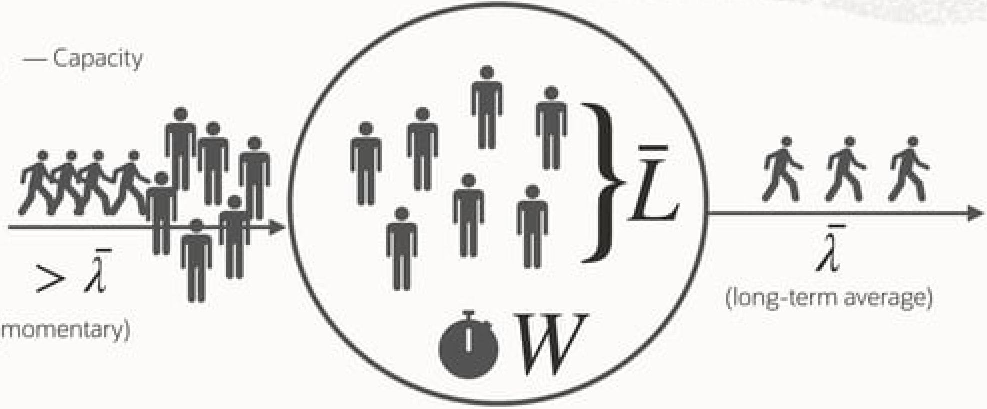


$$\bar{L} = \bar{\lambda} W$$

Copyright © 2022, Oracle and/or its affiliates

Little's Law

$\bar{L}$ — Capacity

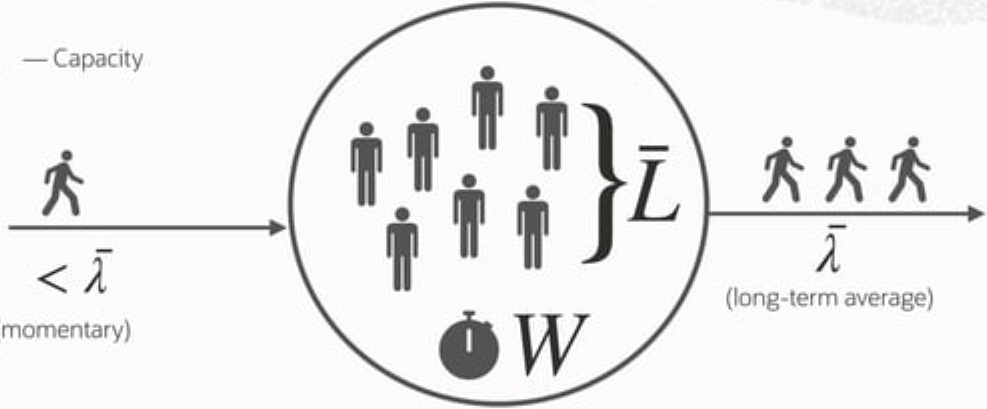


$$\bar{L} = \bar{\lambda} W$$

Copyright © 2022, Oracle and/or its affiliates

Little's Law

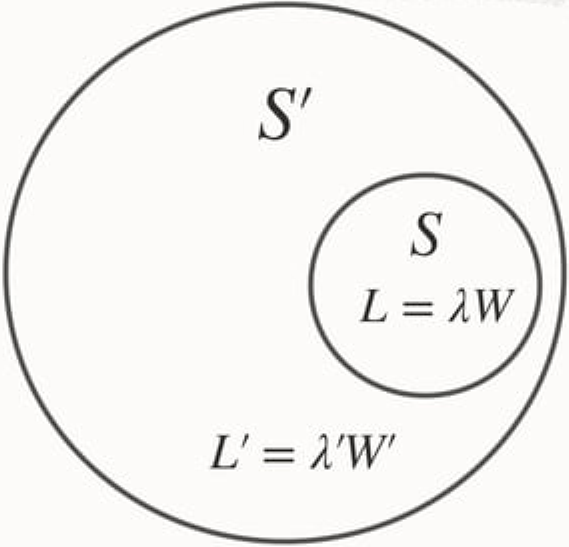
$\bar{L}$ — Capacity



$$\bar{L} = \bar{\lambda} W$$

Copyright © 2022, Oracle and/or its affiliates

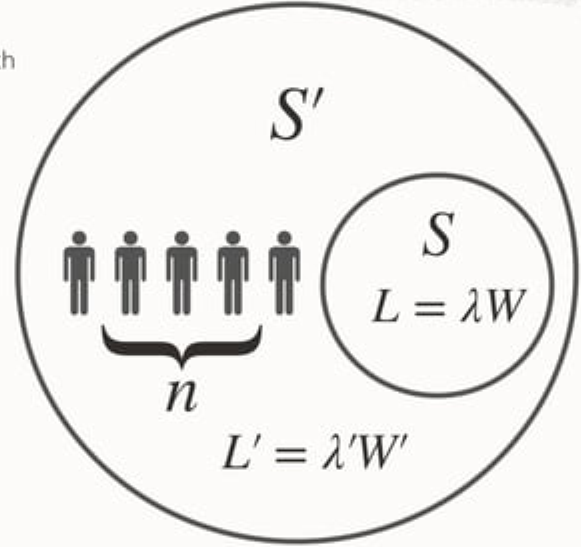
Little's Law — subsystems



Copyright © 2022, Oracle and/or its affiliates

Little's Law — subsystems and queues

n — Average queue length



Copyright © 2022, Oracle and/or its affiliates

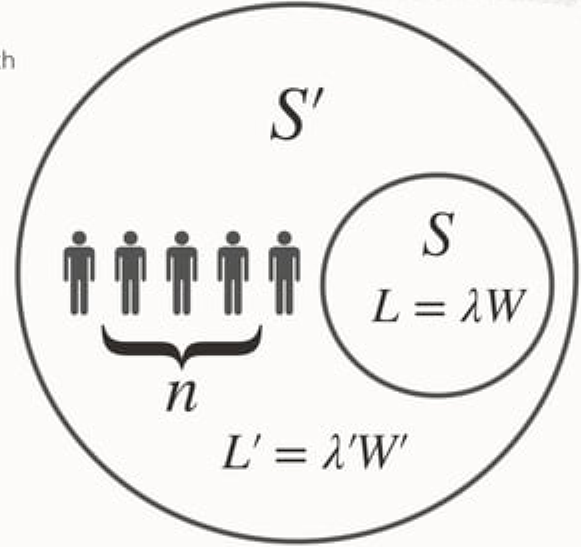
Little's Law — subsystems and queues

n — Average queue length

$$\lambda' = \lambda$$

$$L' = L + n$$

$$W' = W + n\frac{W}{L}$$



 Copyright © 2022, Oracle and/or its affiliates

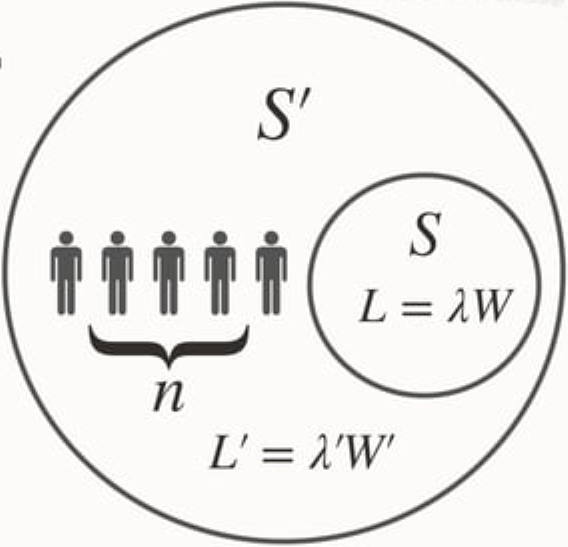
Little’s Law — subsystems and queues

n — Average queue length

$$\lambda' = \lambda$$

$$L' = L + n$$

$$W' = W + n\frac{W}{L}$$



$$L' = \lambda' W'$$

$$L + n = \lambda(W + n\frac{W}{L})$$

$$L\cancel{(1 + \frac{n}{L})} = \lambda W\cancel{(1 + \frac{n}{L})}$$

$$L = \lambda W$$

Copyright © 2022, Oracle and/or its affiliates

Little's Law — other applications

Caching

$P(\text{cache hit}) = p$

$$L_1 = p\lambda W_1$$

$P(\text{cache miss}) = 1 - p$

$$L_2 = (1-p)\lambda W_2$$

$$L = L_1 + L_2 = \lambda(\underbrace{pW_1 + (1-p)W_2}_{W})$$

Interference

x — Interference coefficient

$$L = \lambda(1 + xL)W$$

$$L = \frac{\lambda W}{1 - x\lambda W}$$

$$= \lambda W + (x\lambda W)^2 + (x\lambda W)^3 + \ldots$$

Copyright © 2022, Oracle and/or its affiliates

Capacity and throughput

$\bar{\lambda}$ — Maximum throughput

$\bar{L}$ — Capacity

W — Average latency

$$\bar{\lambda} = \frac{\bar{L}}{W}$$

Copyright © 2022, Oracle and/or its affiliates

CPU capacity

$\bar{\lambda}$ — Maximum throughput

$\bar{L}_{CPU}$ — #cores

W_{CPU} — Average CPU consumption

$$\bar{\lambda} = \frac{\bar{L}_{CPU}}{W_{CPU}}$$

$\bar{L} = 30$ cores

$W_{CPU} = 100\mu s = 1 \times 10^{-4} s$ (> 1500 cache misses)

$\bar{\lambda} = 30{,}000$ req/s

Copyright © 2022, Oracle and/or its affiliates

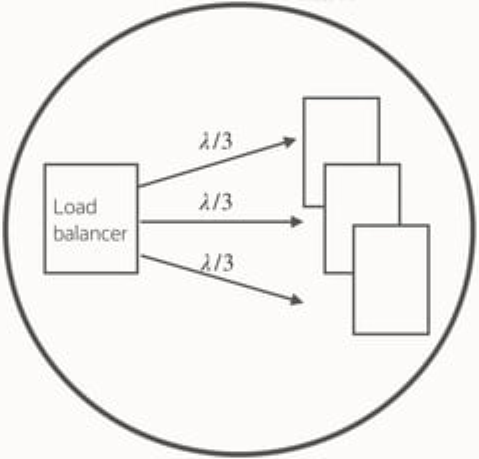
Capacity and throughput

$\bar{\lambda}$ — Maximum throughput

$\bar{L}$ — Capacity

W — Average latency

$$\bar{\lambda} = \frac{\bar{L}}{W}$$



Copyright © 2022, Oracle and/or its affiliates

Thread-per-Request — Thread Capacity

Request: The domain's unit of concurrency
Thread: The software unit of concurrency

Thread-per-Request: A request consumes a thread
(either new or borrowed from a pool) for its duration, *W*

 Copyright © 2022, Oracle and/or its affiliates

Thread-per-Request — Thread Capacity

Request: The domain's unit of concurrency

Thread: The software unit of concurrency

#threads $= L = \lambda W$

Thread-per-Request: A request consumes a thread (either new or borrowed from a pool) for its duration, W

Copyright © 2022, Oracle and/or its affiliates

Thread-per-Request — Thread Capacity

Request: The domain's unit of concurrency

Thread: The software unit of concurrency

$$\#\text{threads} = L = \lambda W$$

Thread-per-Request: A request consumes a thread (either new or borrowed from a pool) for its duration, *W*

with parallel fanout

c — average fanout, i.e. average number of threads consumed by a request

$$\#\text{threads} = \cancel{c}L = \lambda(\frac{W}{\cancel{c}}) = \lambda W$$

⇒ *For the purpose of estimating #threads*, we can consider W to be the sum of all fanout latencies, even if they're done in parallel.

Copyright © 2022, Oracle and/or its affiliates

CPU vs. Thread Capacity with thread/req

$$\bar{L} = \bar{\lambda} W$$
$$= \left(\frac{\bar{L}_{CPU}}{W_{CPU}}\right) W$$
$$= \bar{L}_{CPU}\left(\frac{W}{W_{CPU}}\right)$$

$$\bar{L}_{CPU} = \text{\#cores}$$

$W_{CPU} = 100\mu s$
(> 1500 cache misses) $\Rightarrow$ 100 threads per core
$W = 10ms$

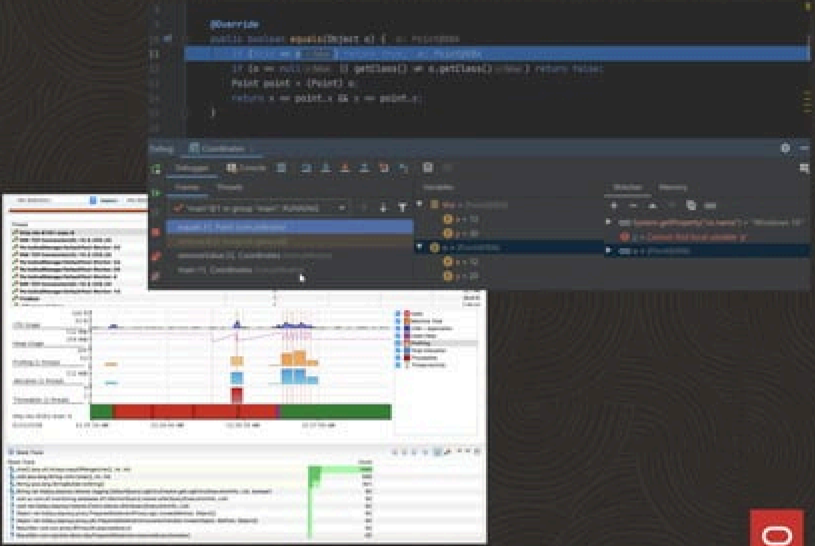
$W_{CPU} = 50\mu s$
(> 800 cache misses) $\Rightarrow$ 1000 threads per core
$W = 50ms$

$W_{CPU} = 10\mu s$
(> 150 cache misses) $\Rightarrow$ 10,000 threads per core
$W = 100ms$

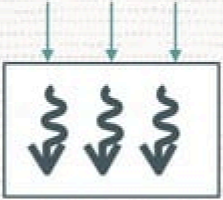
Copyright © 2022, Oracle and/or its affiliates

Java Is Made of Threads

- Exceptions
- Thread Locals
- Debugger
- Profiler (JFR)



 Copyright © 2022, Oracle and/or its affiliates

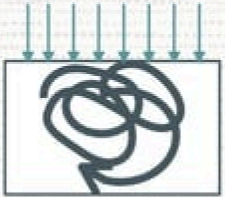


simple
less scalable

SYNC

Programmer 😀
Hardware ☹️

OR



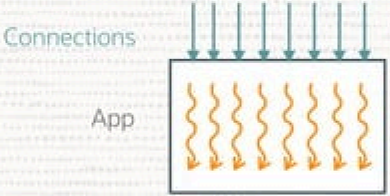
scalable,
complex,
non-interoperable,
hard to debug/profile

ASYNC

Programmer ☹️
Hardware 😀

Copyright © 2022, Oracle and/or its affiliates

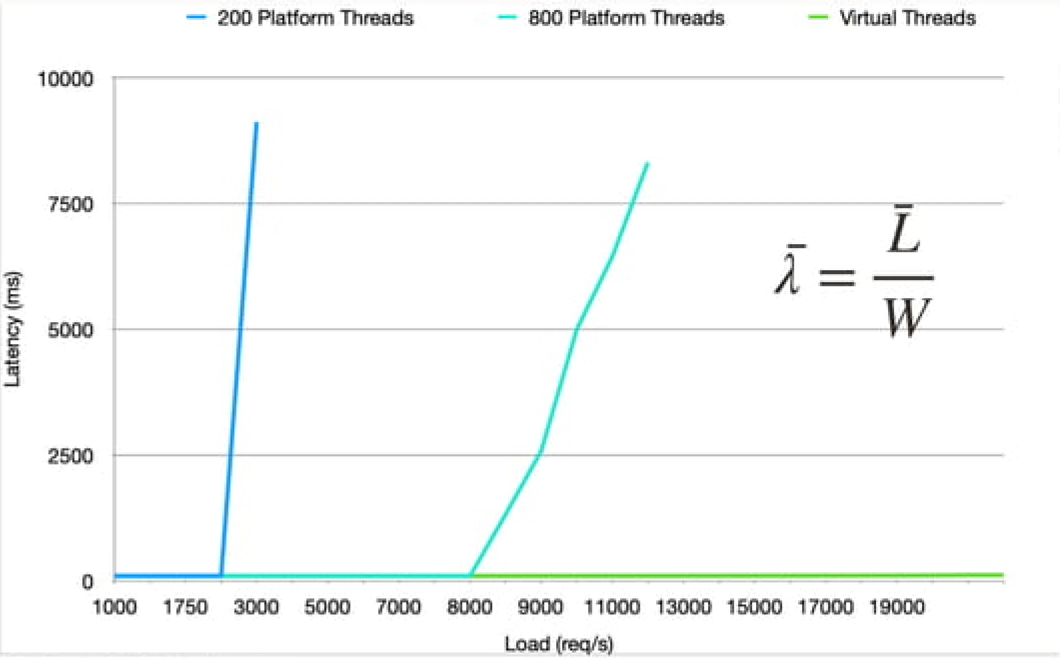
Virtual Threads



Programmer 😀

Hardware 😀

 Copyright © 2022, Oracle and/or its affiliates



Copyright © 2022, Oracle and/or its affiliates

The Impact of Context Switching

- Context-switching affects throughput by means of duration, not capacity $(\frac{\lambda_1}{\lambda_2} = \frac{W_2}{W_1})$

$$W = n(\mu + t) \quad \text{where} \quad \begin{array}{ll} n & \text{avg. \#operations} \\ \mu & \text{avg. wait duration} \\ t & \text{avg. context-switch duration} \end{array}$$

$$\frac{n(\mu + t)}{n\mu} = 1 + \frac{t}{\mu} \quad \Rightarrow \quad \begin{array}{ll} \mu = 20\mu s & \text{(quite fast)} \\ t = 1\mu s & \text{(quite slow)} \end{array} \quad \Rightarrow \quad \text{5\% impact}$$

- Virtual threads have a faster context-switch than OS threads
- Structured concurrency allows waiting for a set of operations with one context-switch

Copyright © 2022, Oracle and/or its affiliates

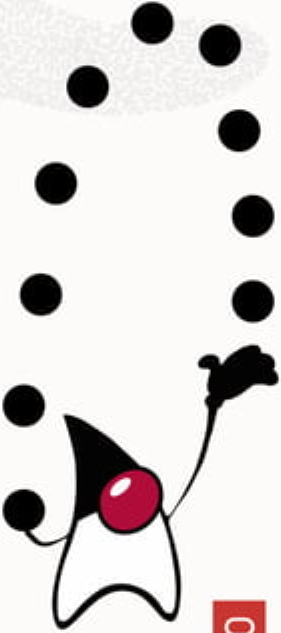
Not cooperative, but no time-sharing yet

- Non-cooperative scheduling is more composable
- … but people overestimate the importance of time-sharing in servers
- Effective time-sharing effective when #threads is #cores×10^5 requires study

Copyright © 2022, Oracle and/or its affiliates

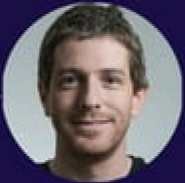
Summary

- Virtual threads allow higher throughput for the thread-per-request style — the style harmonious with the platform — by drastically increasing the request capacity of the server.
- We can juggle more balls not by adding hands but by enlarging the arch.
- Context-switching cost could be important, but aren't the main reason for the throughput increase.



Copyright © 2022, Oracle and/or its affiliates

- JEP 425: Virtual Threads (Preview)
- The Design of User Mode Threads in Java [video] (why not async/await)



Ron Pressler
ron.pressler@oracle.com
@pressron

Brought to you by SCYLLA