Enabling Hybrid Parallel Runtimes Through Kernel and Virtualization Support

Kyle C. Hale and Peter Dinda

Northwestern University

HOBBES

xstack.sandia.gov/hobbes

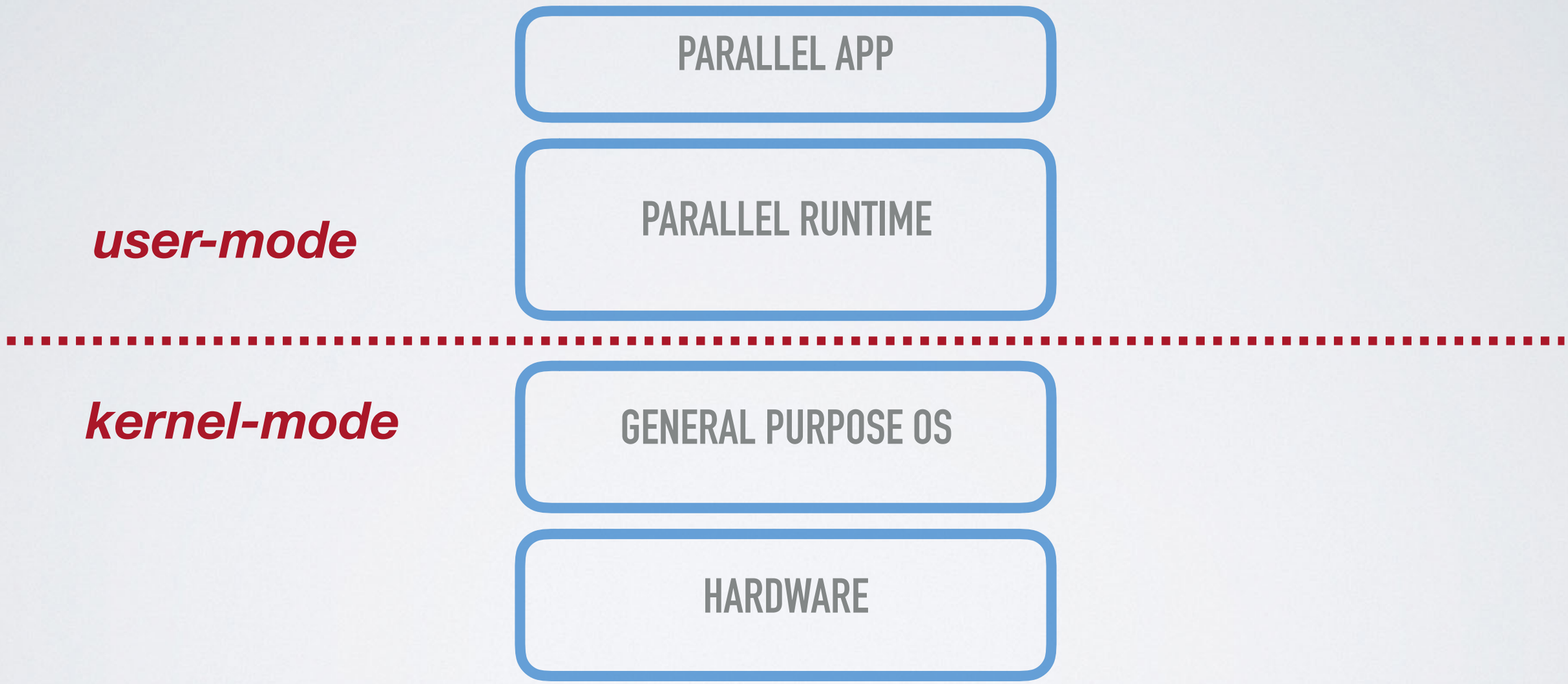
Hybrid Runtimes

the runtime IS the kernel

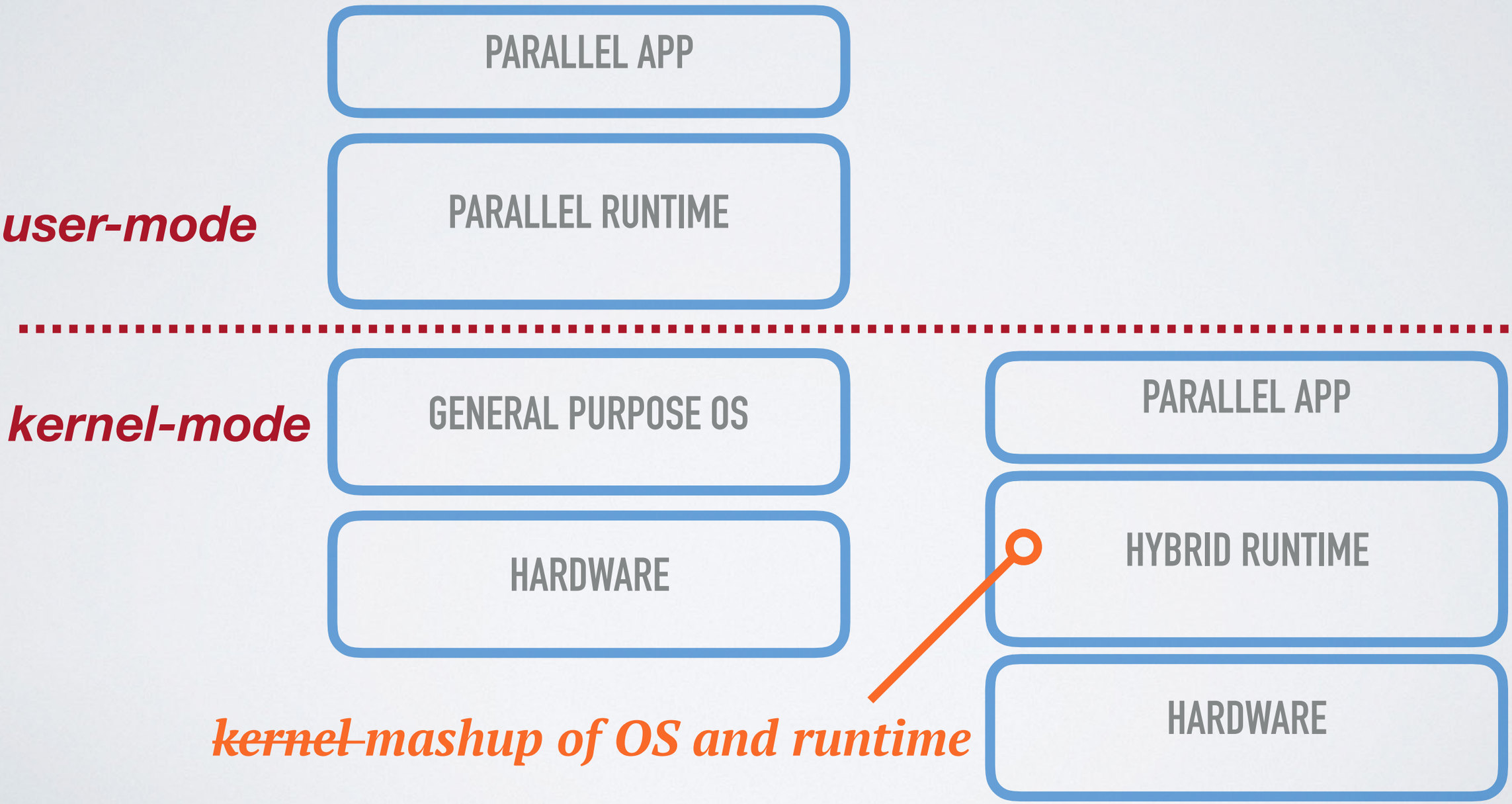
runtime not limited to abstractions exposed by syscall interface

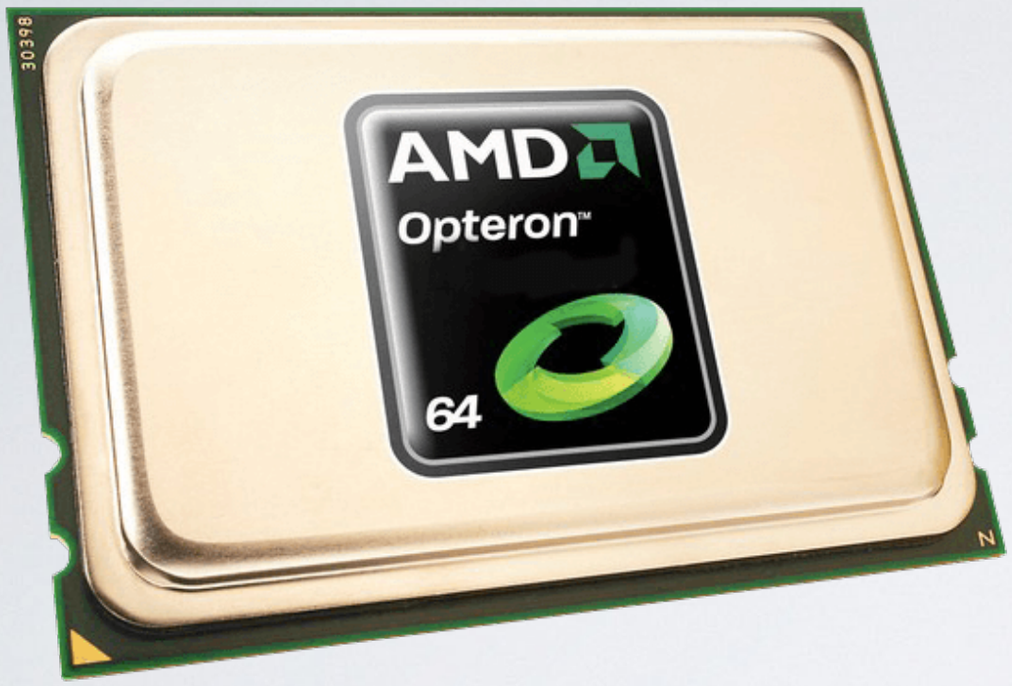
opportunity for leveraging privileged HW features

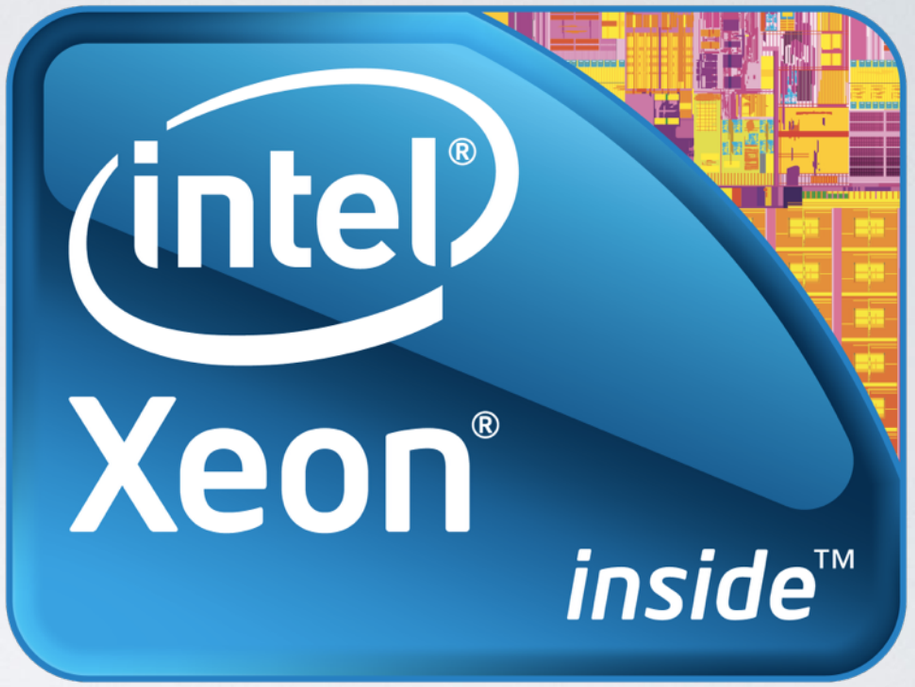
CURRENT OS/R MODEL

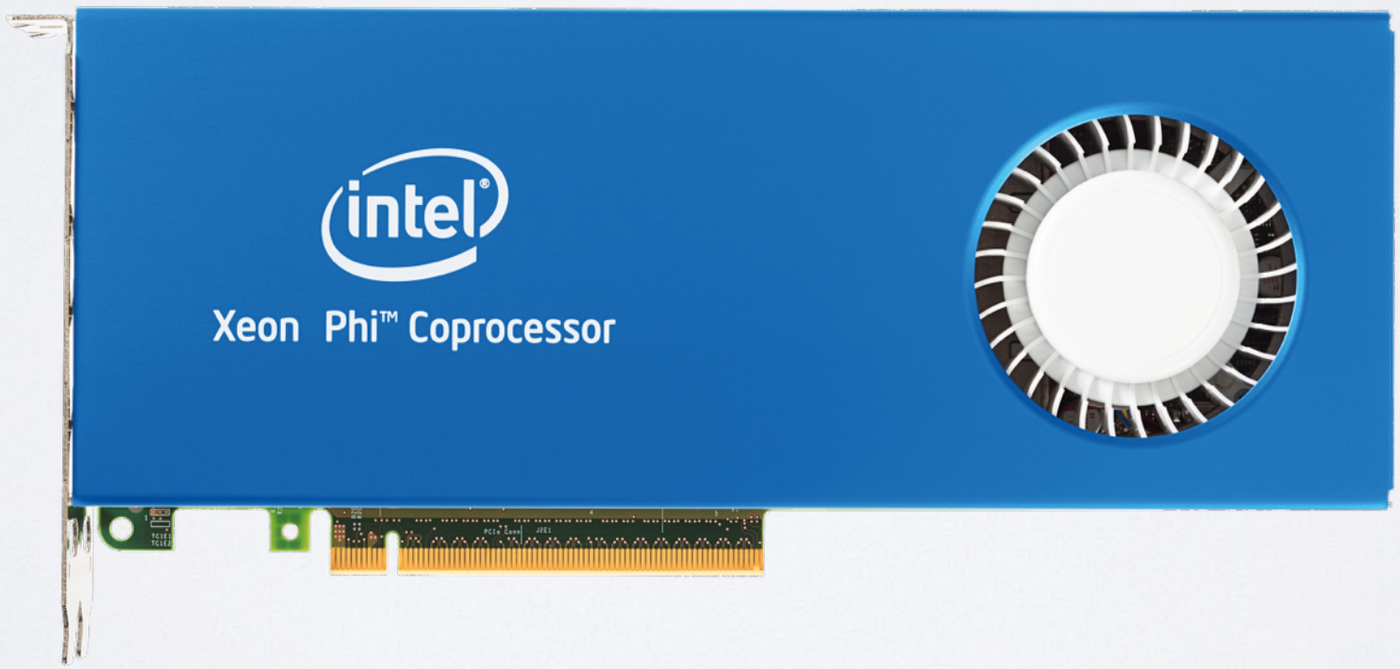


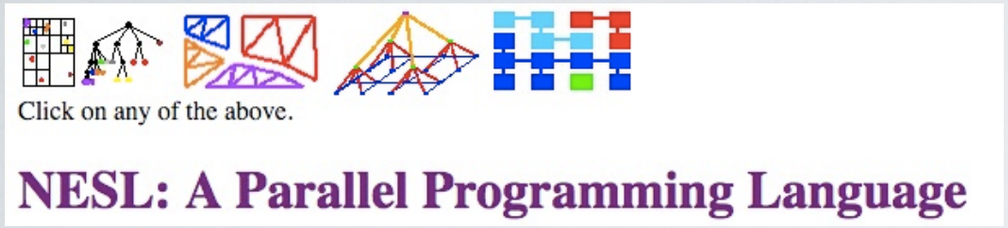
THE HYBRID RUNTIME




AMD
Opteron™
64


intel®
Xeon®
inside™


intel
Xeon Phi™ Coprocessor

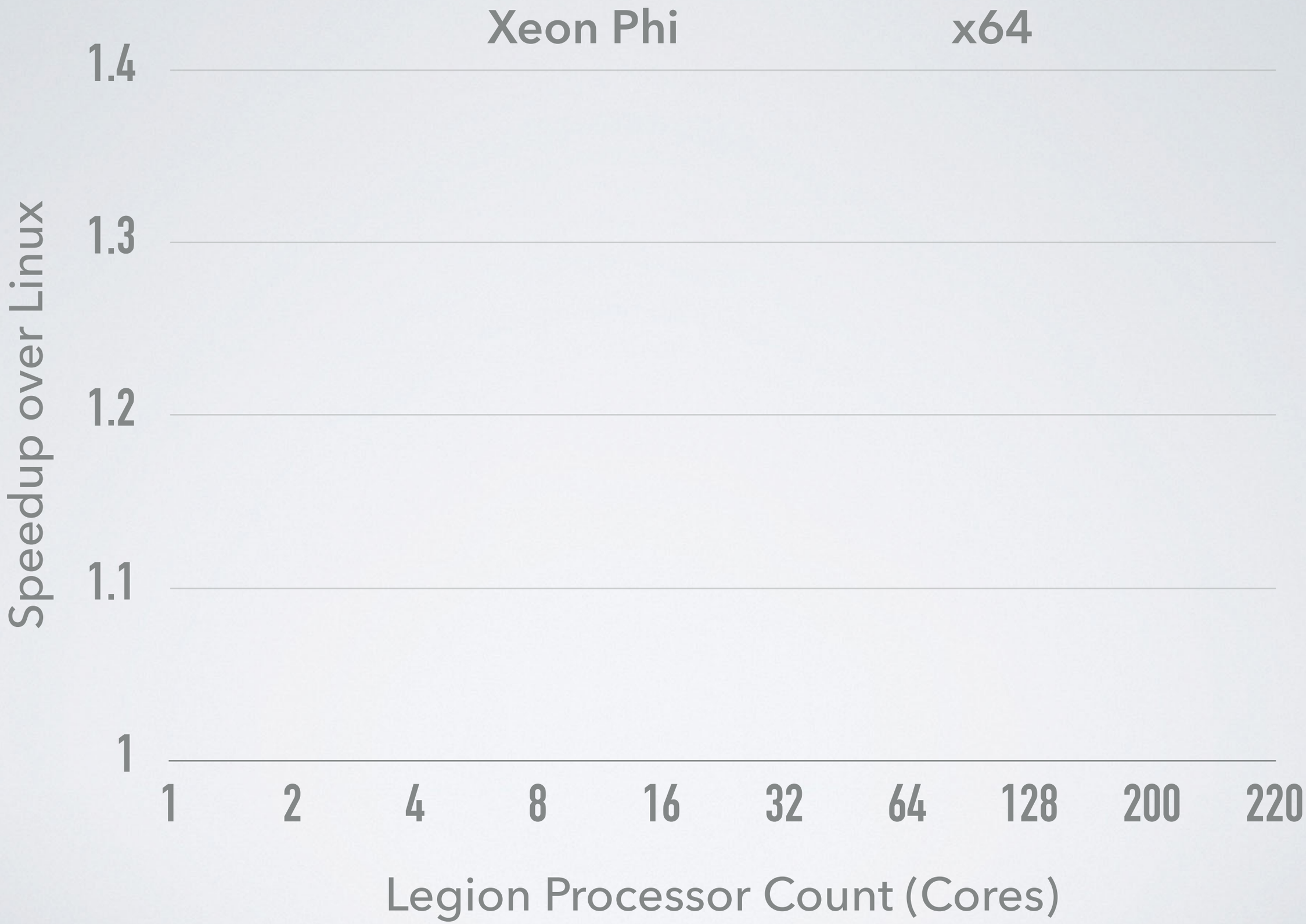


NESL

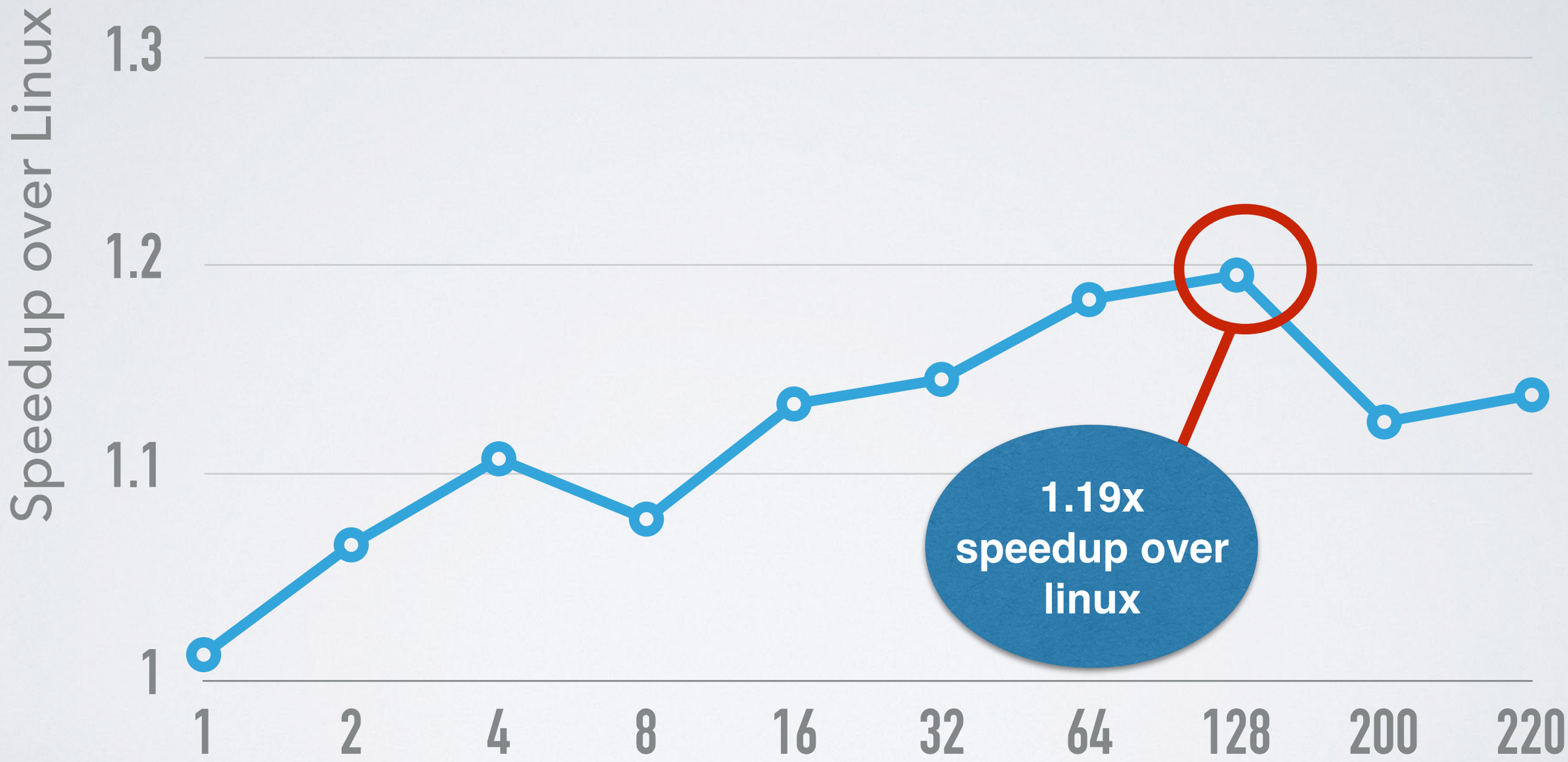
NDPC

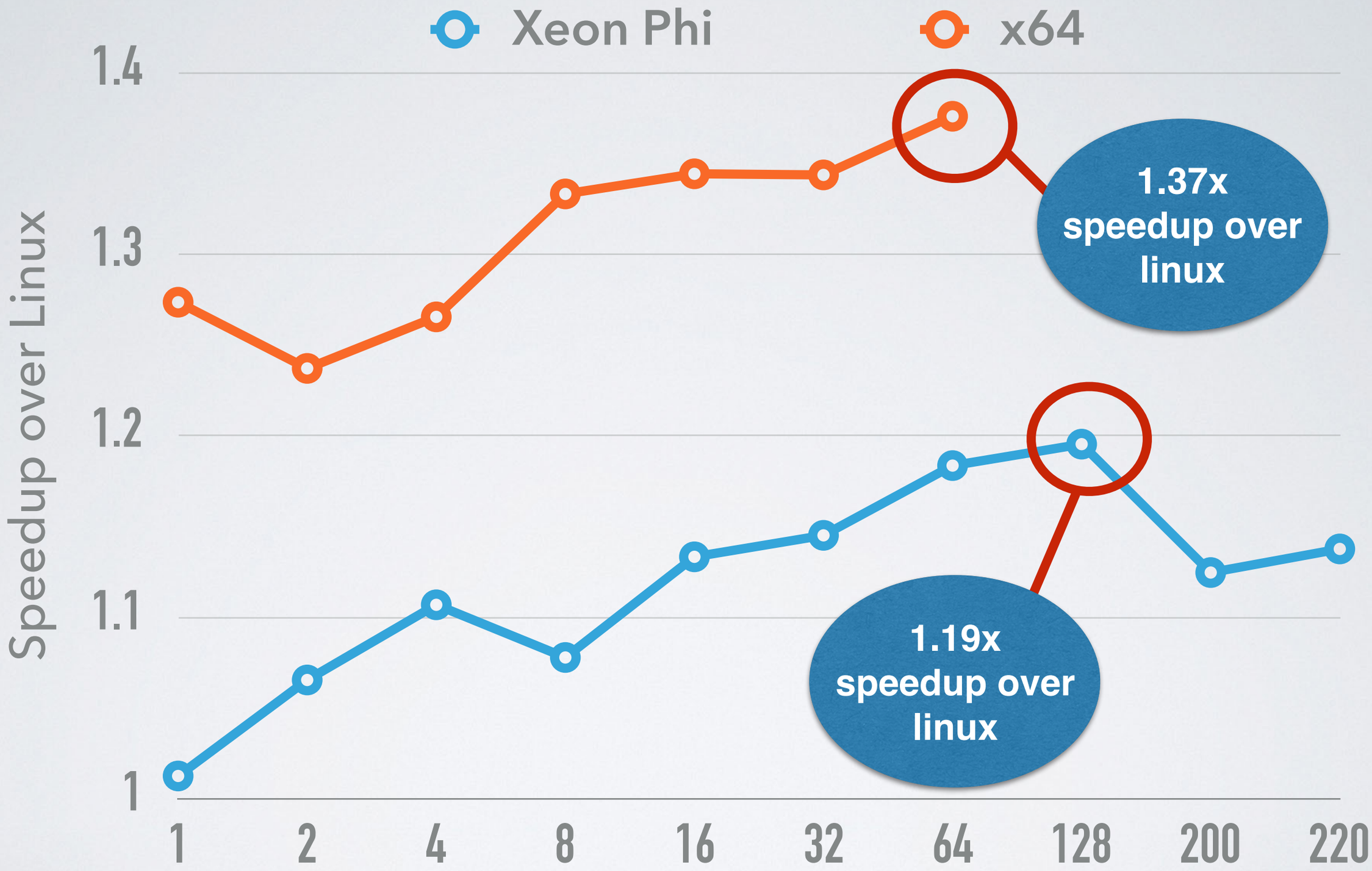
Racket

Legion


Xeon Phi
x64
1.4
1.3
1.2
1.1
1
Speedup over Linux
1
2
4
8
16
32
64
128
200
220
Legion Processor Count (Cores)

Xeon Phi x64

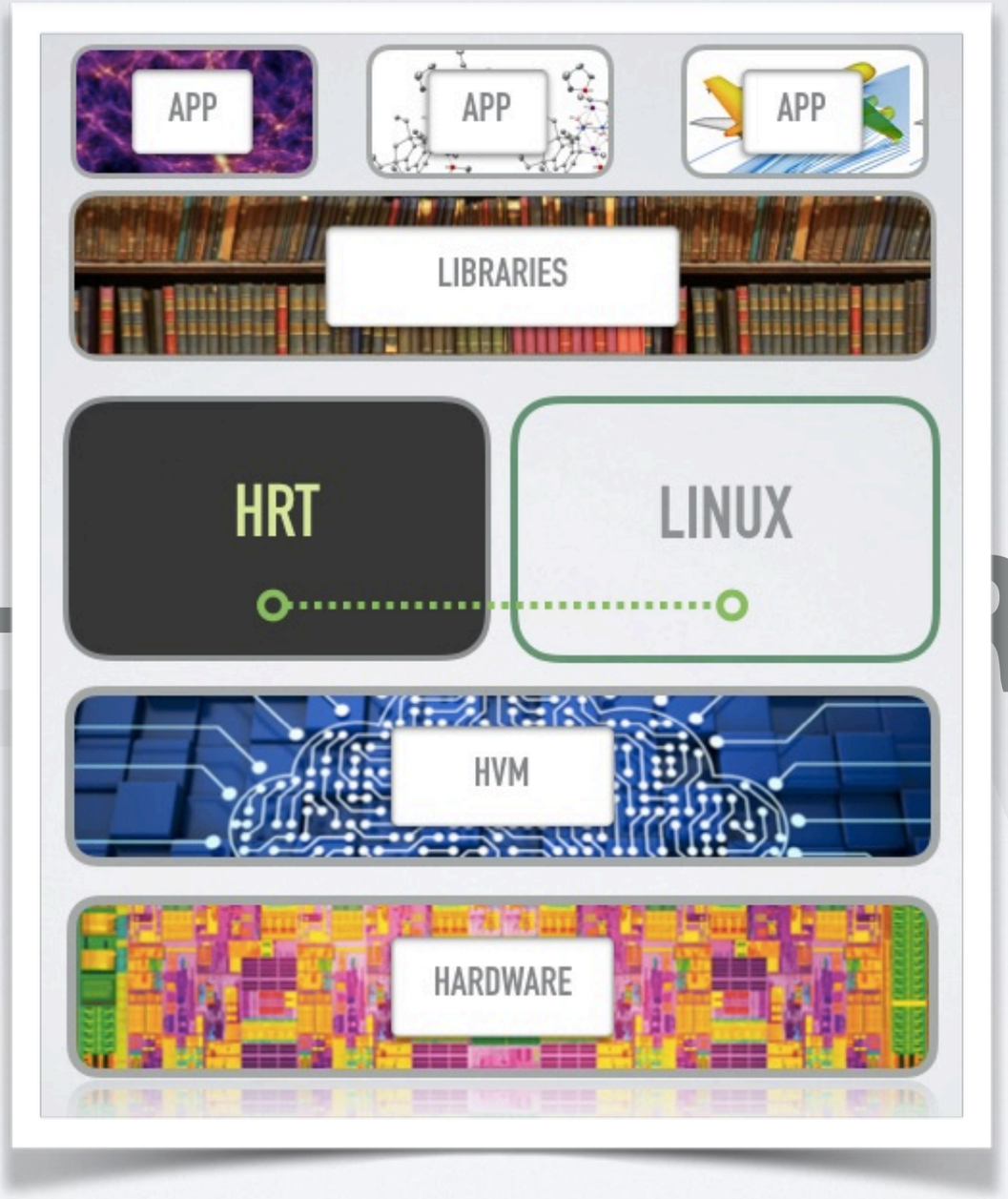

Speedup over Linux
1.4
1.3
1.2
1.1
1
1.19x speedup over linux
1
2
4
8
16
32
64
128
200
220
Legion Processor Count (Cores)


Xeon Phi
x64
1.4
1.3
1.2
1.1
1
Speedup over Linux
1.37x speedup over linux
1.19x speedup over linux
1
2
4
8
16
32
64
128
200
220
Legion Processor Count (Cores)

TWO ENABLING TOOLS

kernel framework

THE H RTUAL



bridge HRT with legacy OS

OUTLINE

Background/Overview

Nautilus

Deployment Models

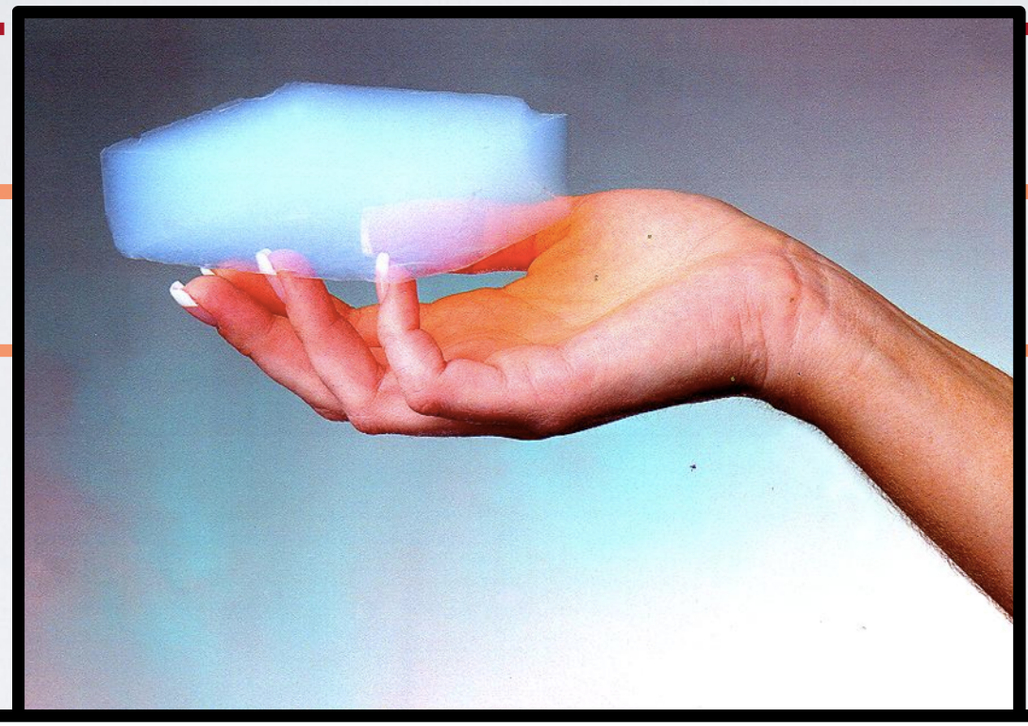
Hybrid Virtual Machine

Multiverse & Future Work

NAUTILUS

user-mode

kernel-mode



aerokernel

HARDWARE

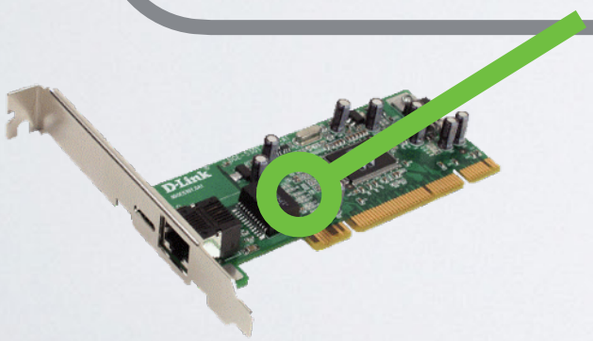
Nautilus primitives & utilities (HRT can use or not use any of them)

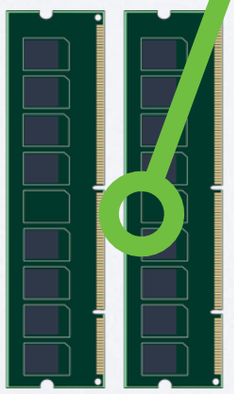
Nautilus

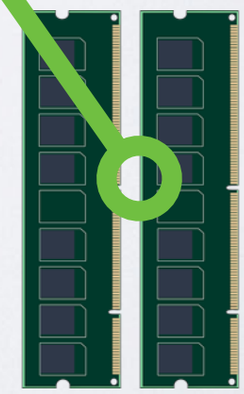
under the hood

Parallel Runtime System

Nautilus







kernel primitives should be SIMPLE and FAST

runtime developer can easily reason about them!

start with familiar interfaces

threads

condition variables

mutexes/locks

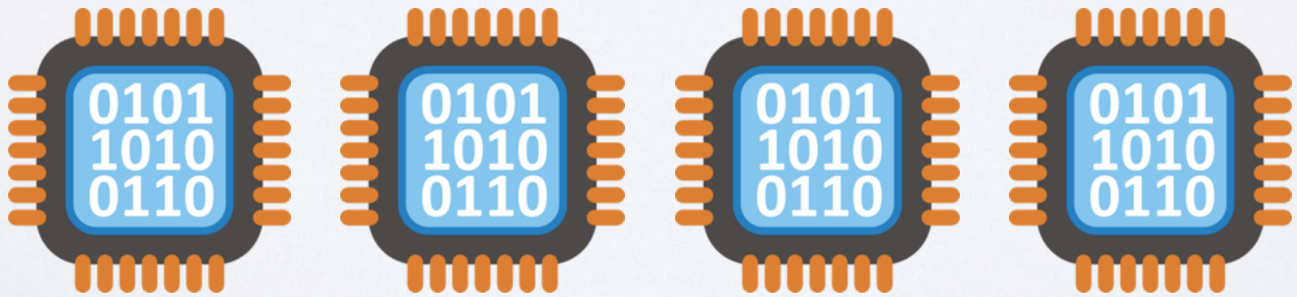
memory management

fork/join parallelism

map runtime's logical view of machine onto physical HW

Parallel Runtime System

logical CPUs



threads

unified, shared address space

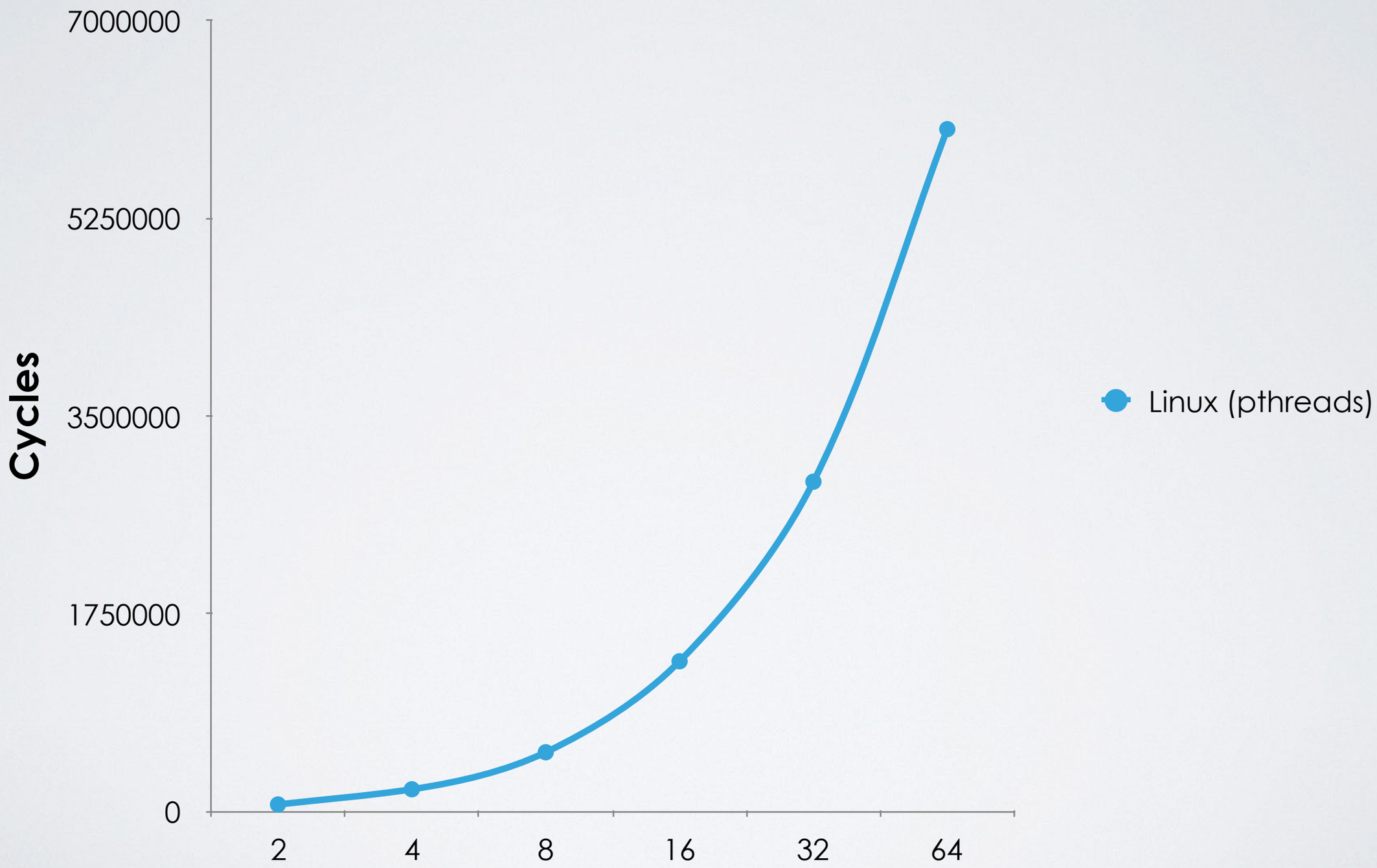
threads can operate preemptively or cooperatively*

*for runtimes that require more determinism

thread creation is **FAST**

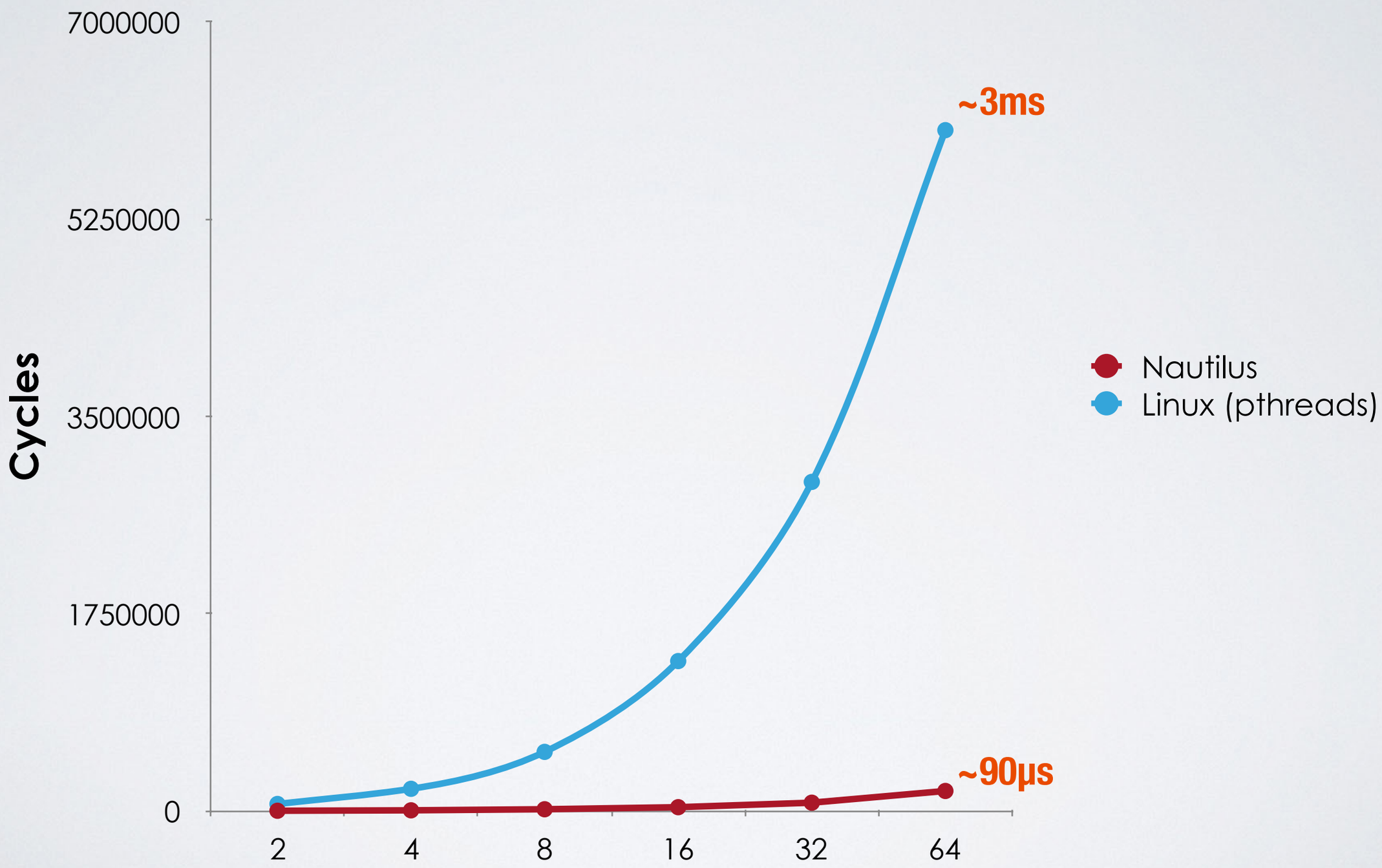
x86_64 Opteron: 64 cores, 4 sockets, 8 numa zones, 128GB RAM

thread creation is **FAST**



x86_64 Opteron: 64 cores, 4 sockets, 8 numa zones, 128GB RAM

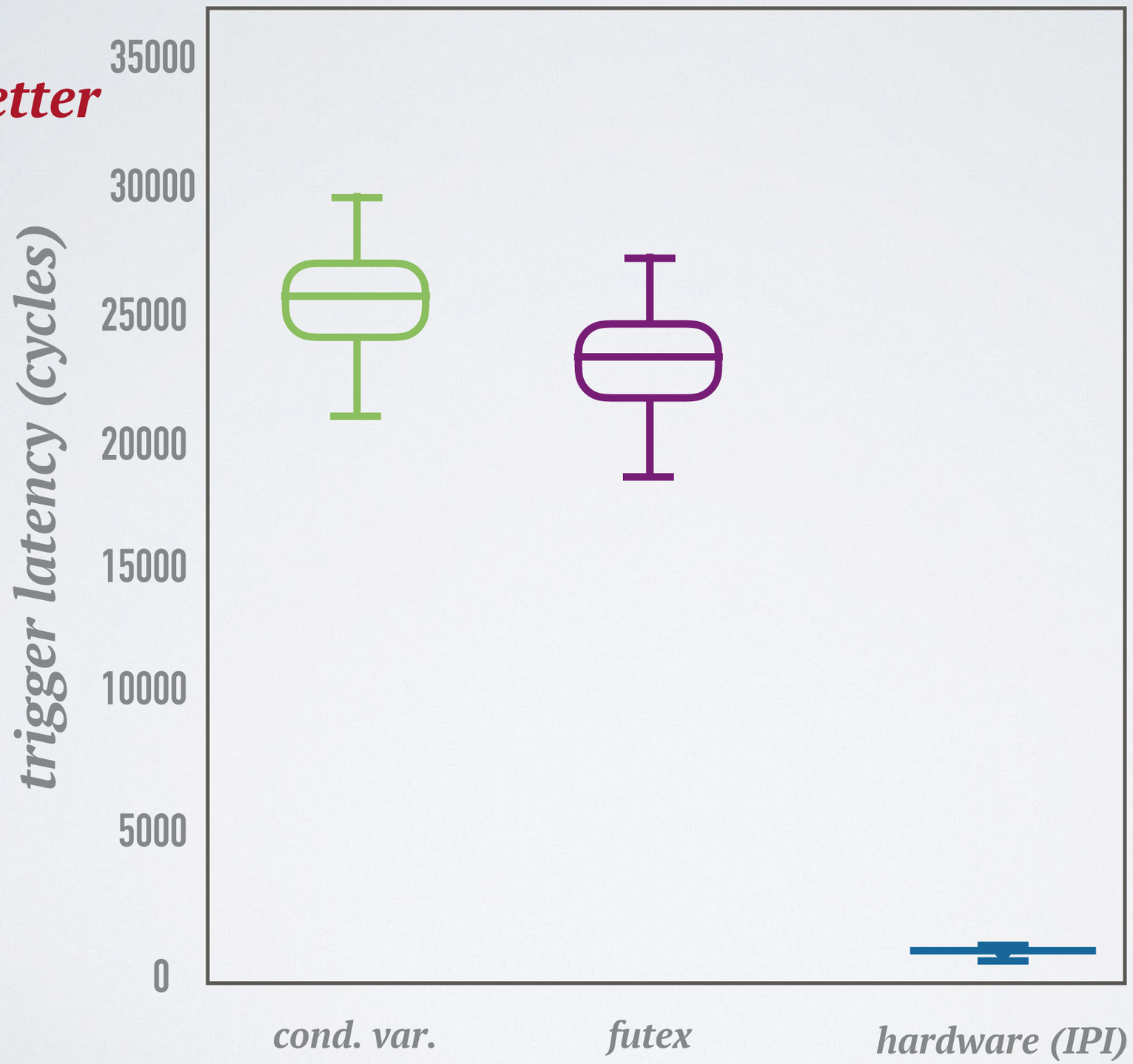
thread creation is **FAST**



x86_64 Opteron: 64 cores, 4 sockets, 8 numa zones, 128GB RAM

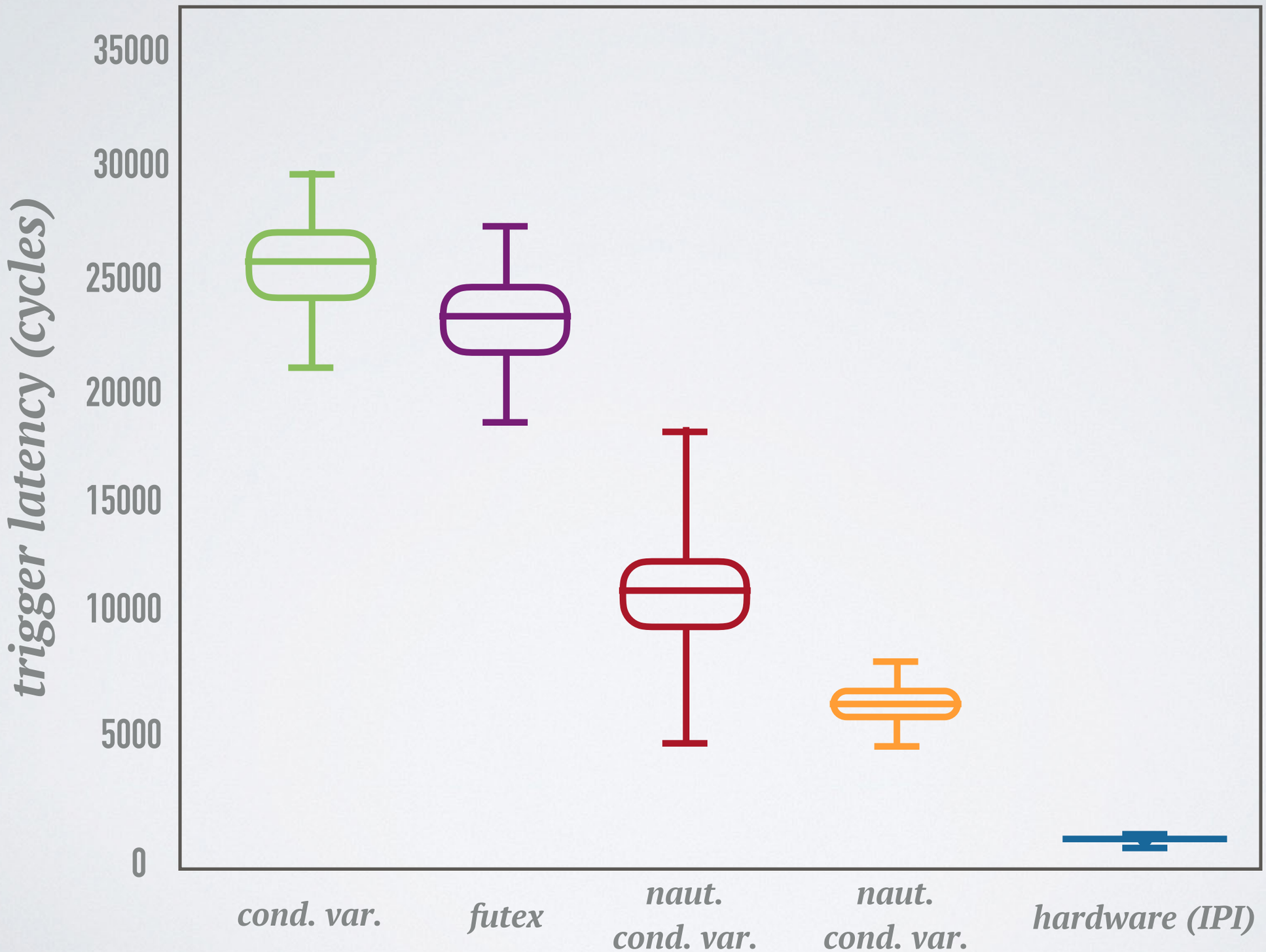
software events *user-mode software events are* **SLOW**

lower is better

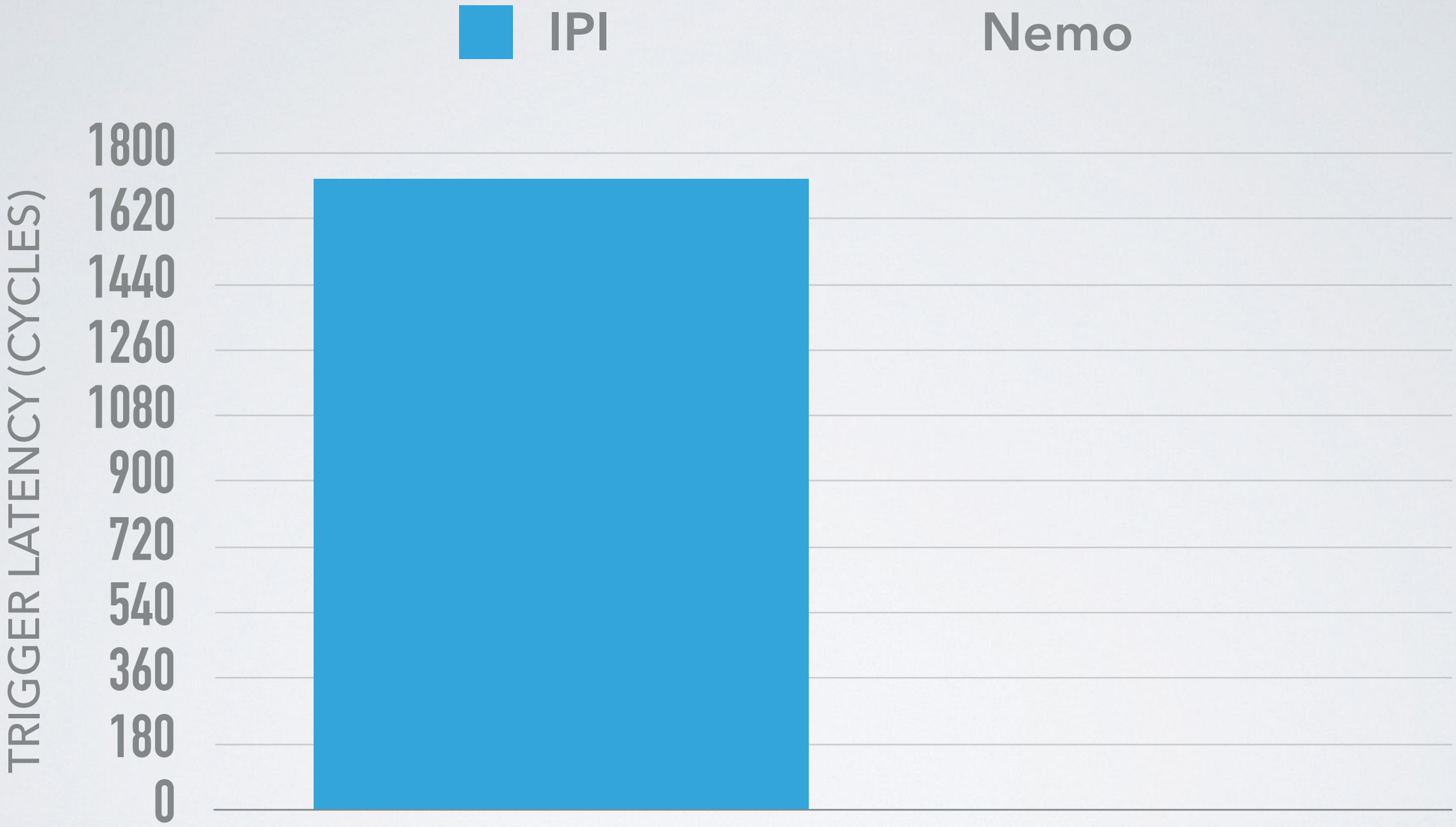


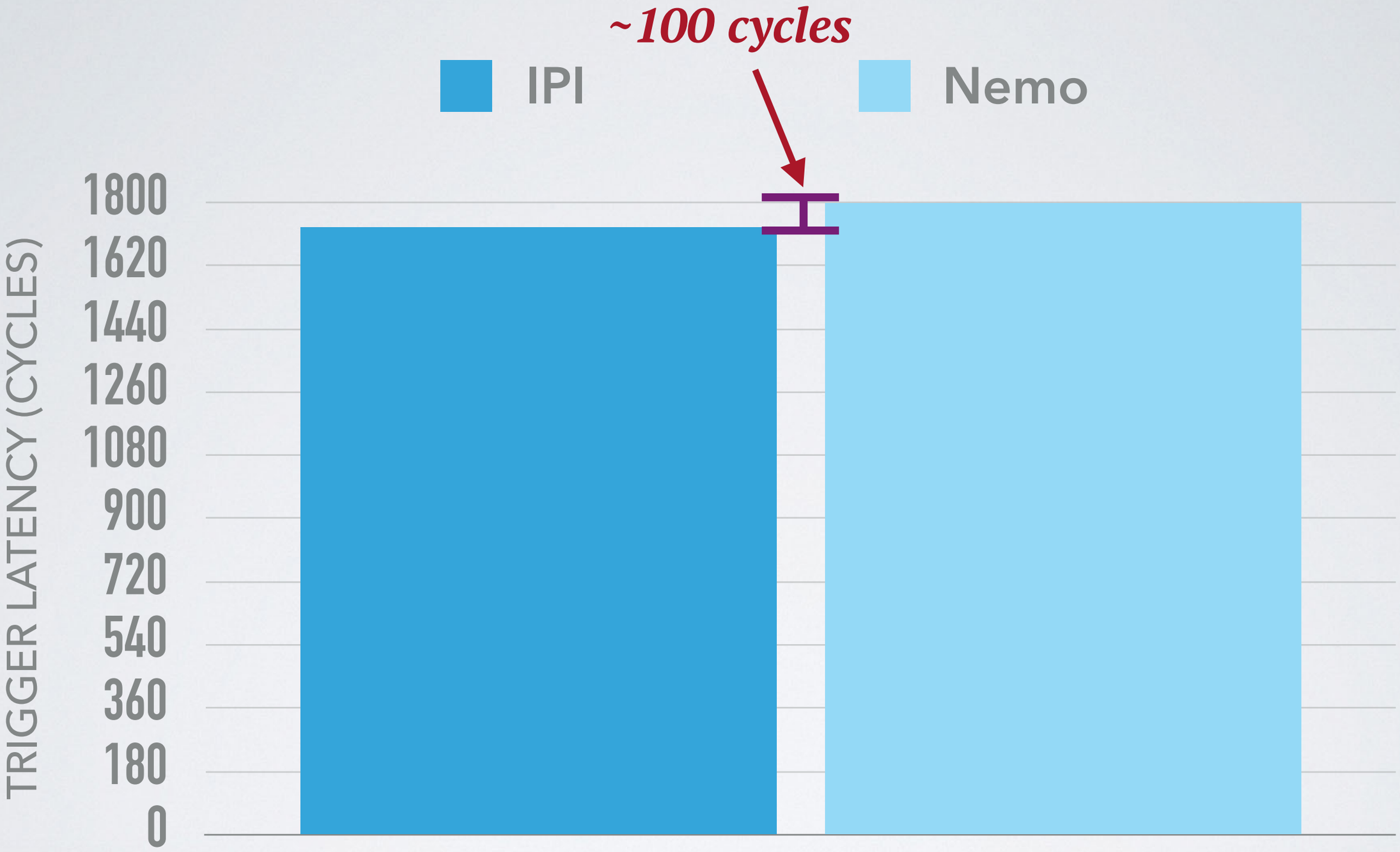
x86_64 Opteron: 64 cores, 4 sockets, 8 numa zones, 128GB RAM

nautilus events triggers are **FASTER**



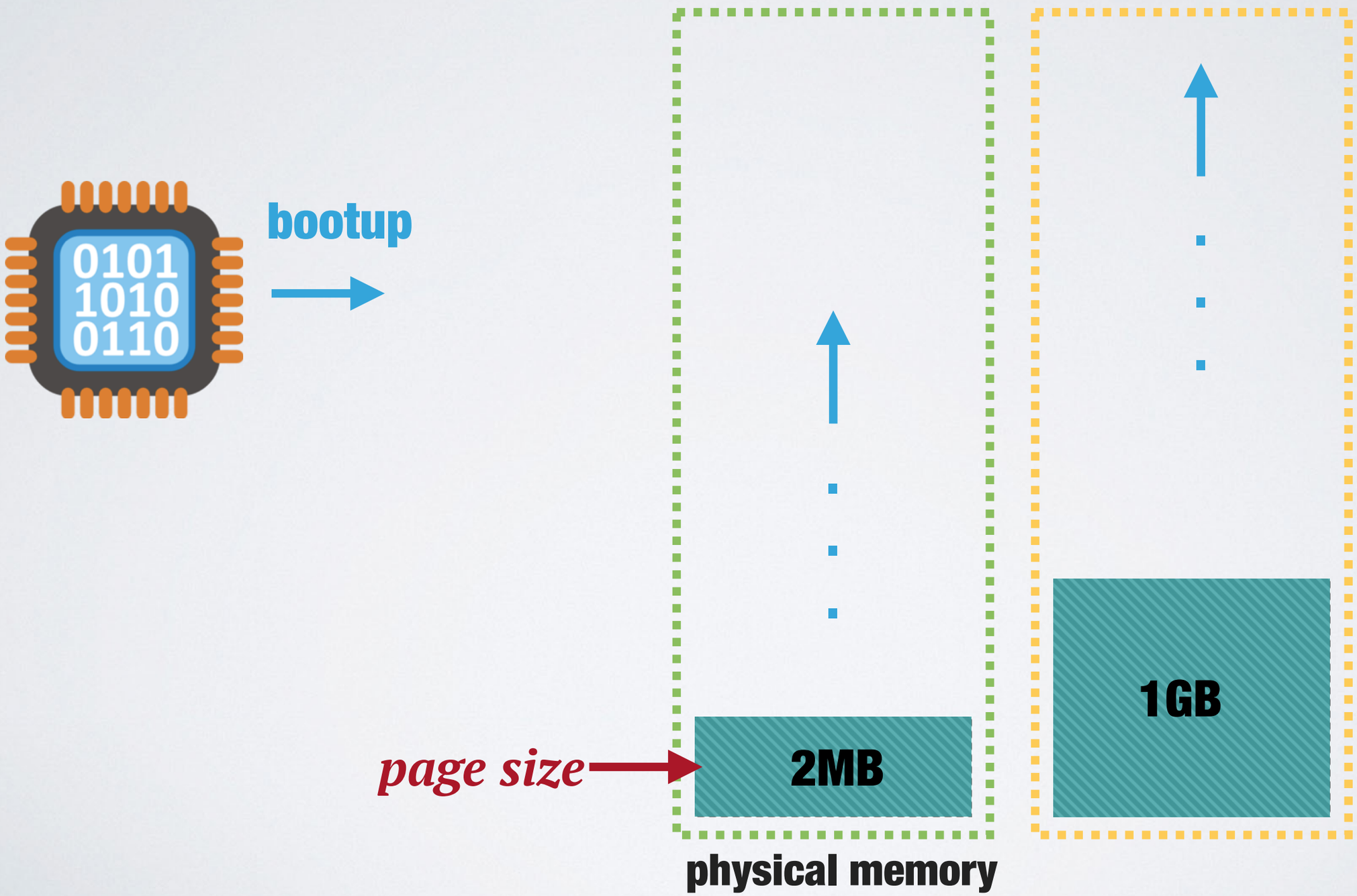
x86_64 Opteron: 64 cores, 4 sockets, 8 numa zones, 128GB RAM


IPI
Nemo
TRIGGER LATENCY (CYCLES)
1800
1620
1440
1260
1080
900
720
540
360
180
0


~100 cycles
IPI
Nemo
TRIGGER LATENCY (CYCLES)
1800
1620
1440
1260
1080
900
720
540
360
180
0

memory

static identity map



eliminates expensive page faults

reduces TLB misses + shootdowns

increases performance under virtualization

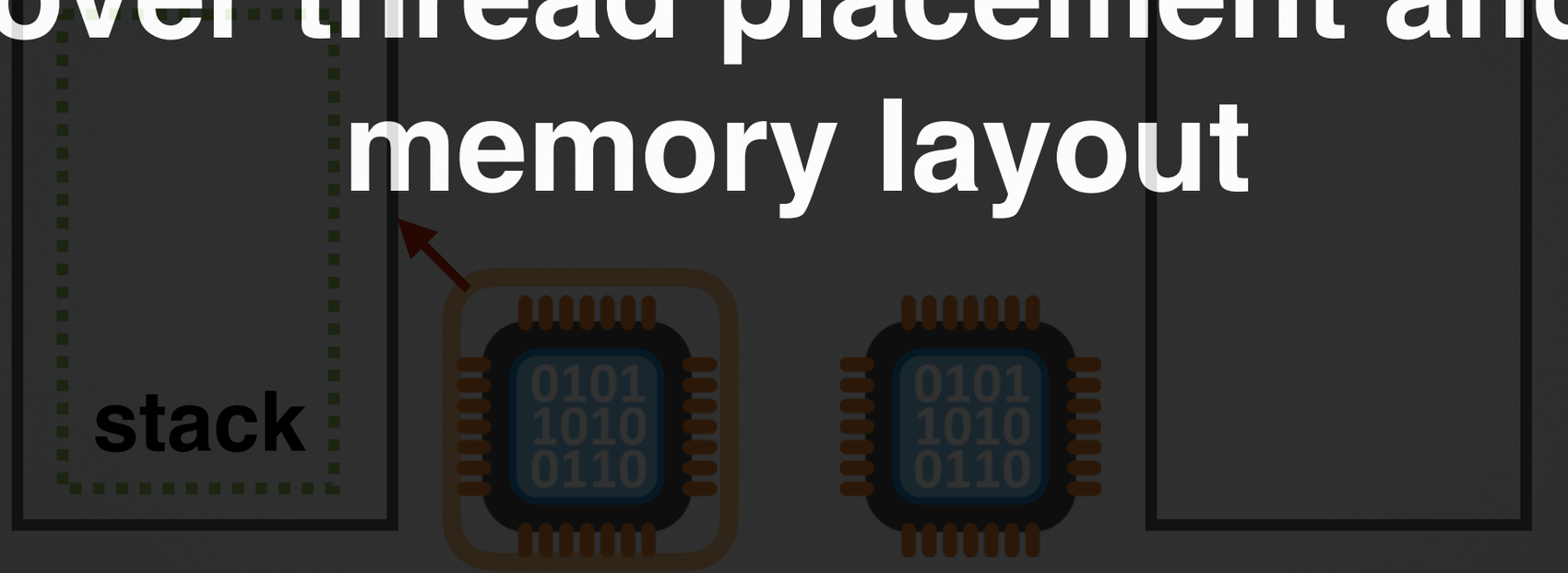
NUMA

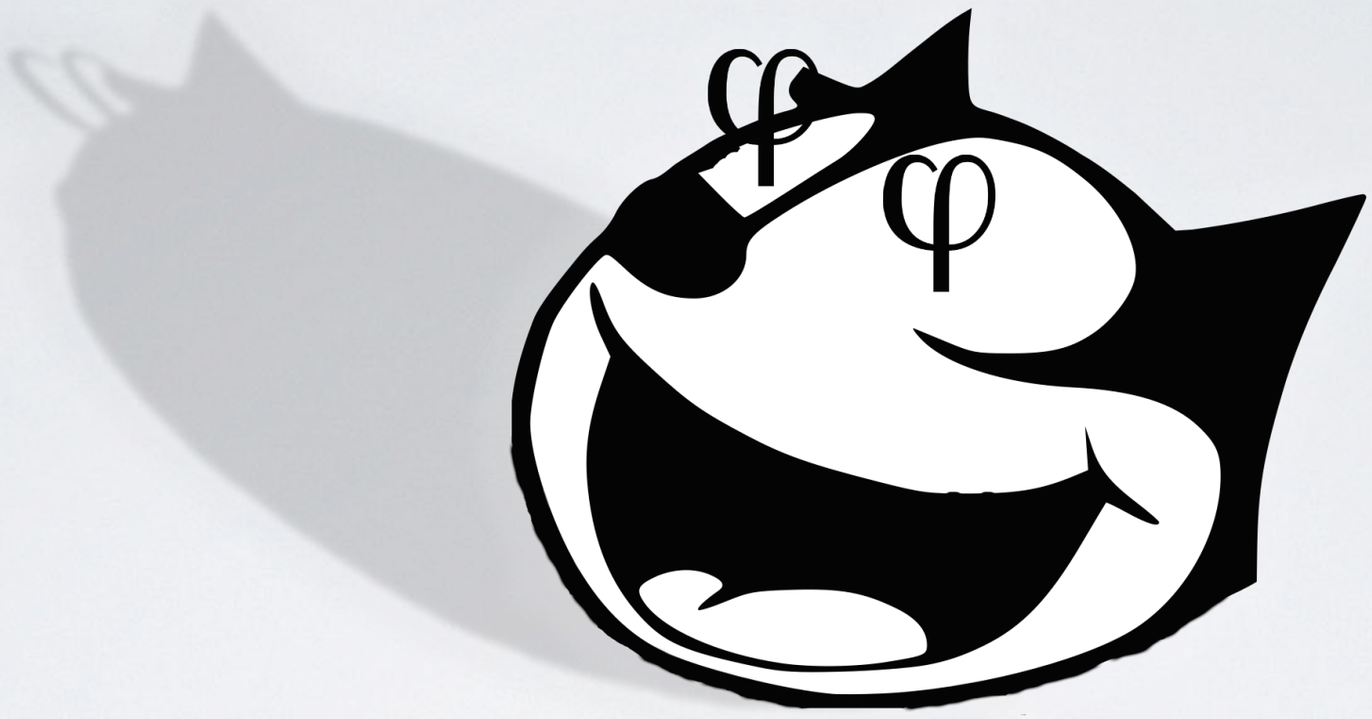
~~first-touch~~

NUMA domain

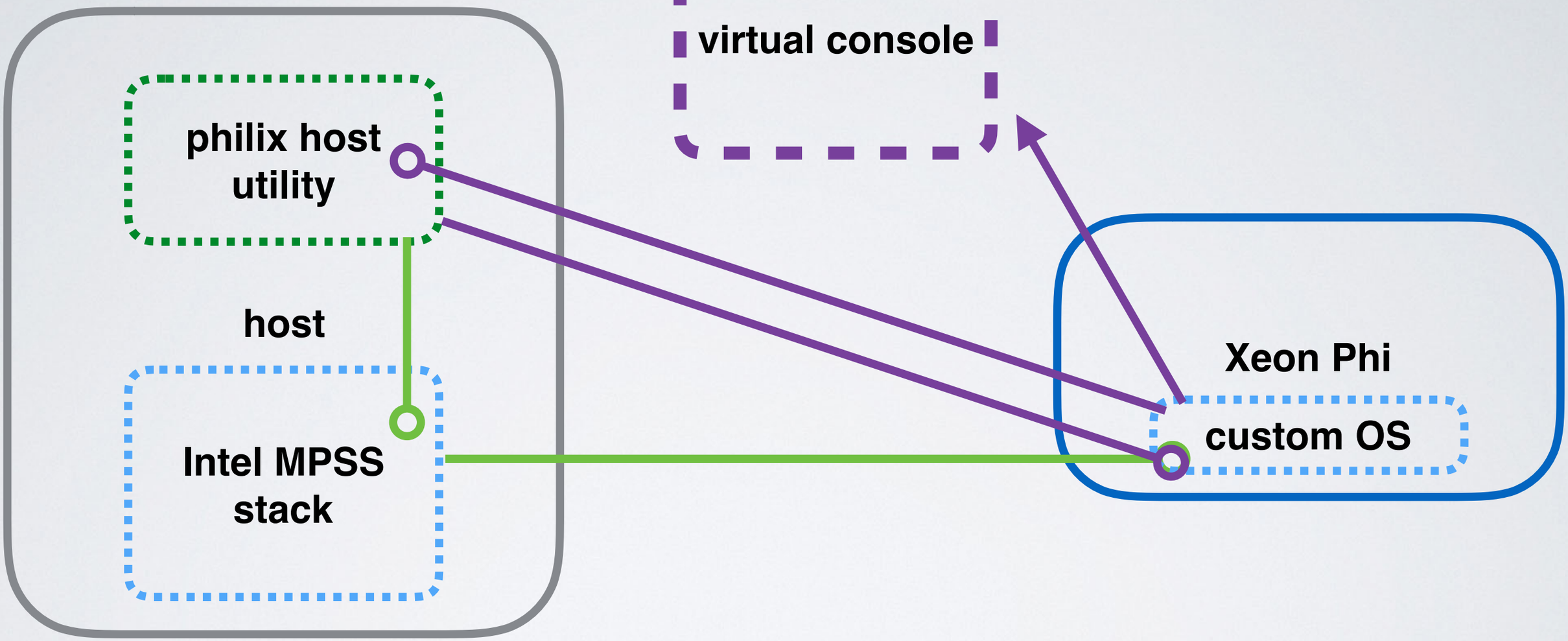
NUMA domain

runtime has FULL control over thread placement and memory layout





philix


virtual console
philix host utility
host
Intel MPSS stack
Xeon Phi
custom OS

Nautilus is fairly small

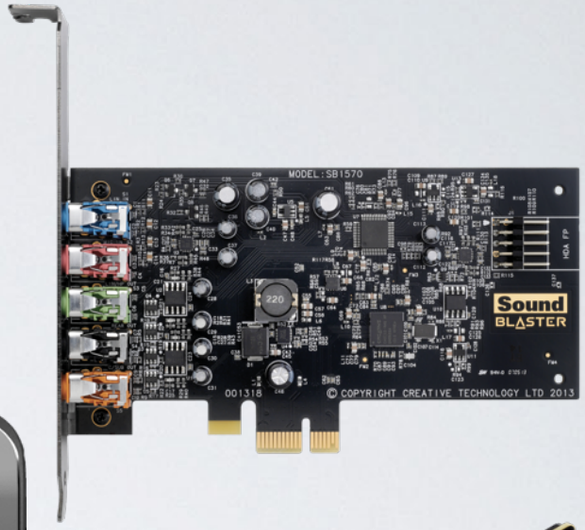
Nautilus	25,000 lines of mostly C
Legion	43,000 lines of C++
Additions for Legion	800 lines of C/C++
Additions for Xeon Phi	1350 lines of C

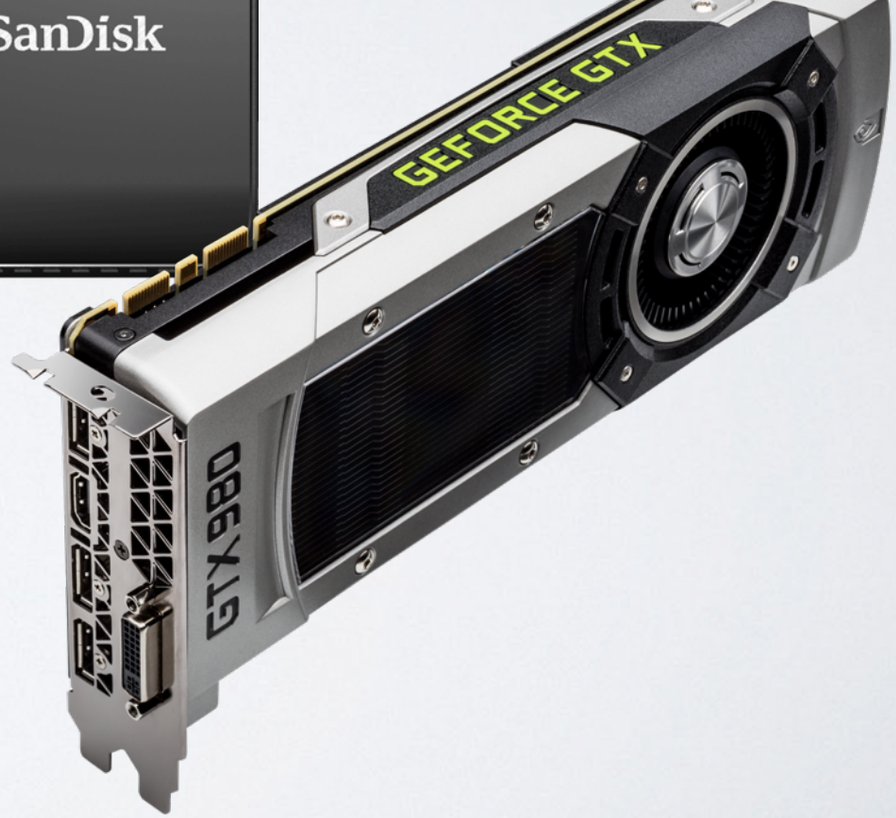
What do we lose?

```
[root@localhost ~]# ping -q fa.wikipedia.org
PING text.pmtpa.wikimedia.org (208.80.152.2) 56(84) bytes of data.
^C
--- text.pmtpa.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 540.528/540.528/540.528/0.000 ms
[root@localhost ~]# pwd
/root
[root@localhost ~]# cd /var
[root@localhost var]# ls -la
total 72
drwxr-xr-x. 18 root root 4096 Jul 30 22:43 .
drwxr-xr-x. 23 root root 4096 Sep 14 20:42 ..
drwxr-xr-x.  2 root root 4096 May 14 00:15 account
drwxr-xr-x. 11 root root 4096 Jul 31 22:26 cache
drwxr-xr-x.  3 root root 4096 May 18 16:03 db
drwxr-xr-x.  3 root root 4096 May 18 16:03 empty
drwxr-xr-x.  2 root root 4096 May 18 16:03 games
drwxr-xr-T.  2 root gdm  4096 Jul  2 18:39 gdm
drwxr-xr-x. 28 root root 4096 May 18 16:03 lib
drwxr-xr-x.  2 root root 4096 May  8  6:03 local
lrwxrwxrwx.  1 root root    4 May 14 00:12 lock -> ../run/lock
drwxr-xr-x. 14 root root 4096 Sep 14 20:42 log
lrwxrwxrwx.  1 root root   10 Jul 30 22:43 mail -> spool/mail
drwxr-xr-x.  2 root root 4096 May 18 16:03 nis
drwxr-xr-x.  2 root root 4096 May 18 16:03 opt
drwxr-xr-x.  2 root root 4096 May 18 16:03 preserve
drwxr-xr-x.  2 root root 4096 Jul  1 22:11 report
lrwxrwxrwx.  1 root root    6 May 14 00:12 run -> ../run
drwxr-xr-x. 14 root root 4096 May 18 16:03 spool
drwxrwxrwt.  4 root root 4096 Sep 12 23:50 tmp
drwxr-xr-x.  2 root root 4096 May 18 16:03 yp
[root@localhost var]# yum search wiki
Loaded plugins: langpacks, presto, refresh-packagekit, remove-with-leaves
rpmfusion-free-updates                                   | 2.7 kB     00:00
rpmfusion-free-updates/primary_db                        | 206 kB     00:04
rpmfusion-nonfree-updates                                | 2.7 kB     00:00
updates/metalink                                         | 5.9 kB     00:00
updates                                                  | 4.7 kB     00:00
updates/primary_db                 73% [==================          ]  62 kB/s | 2.6 MB     00:15 ETA
```

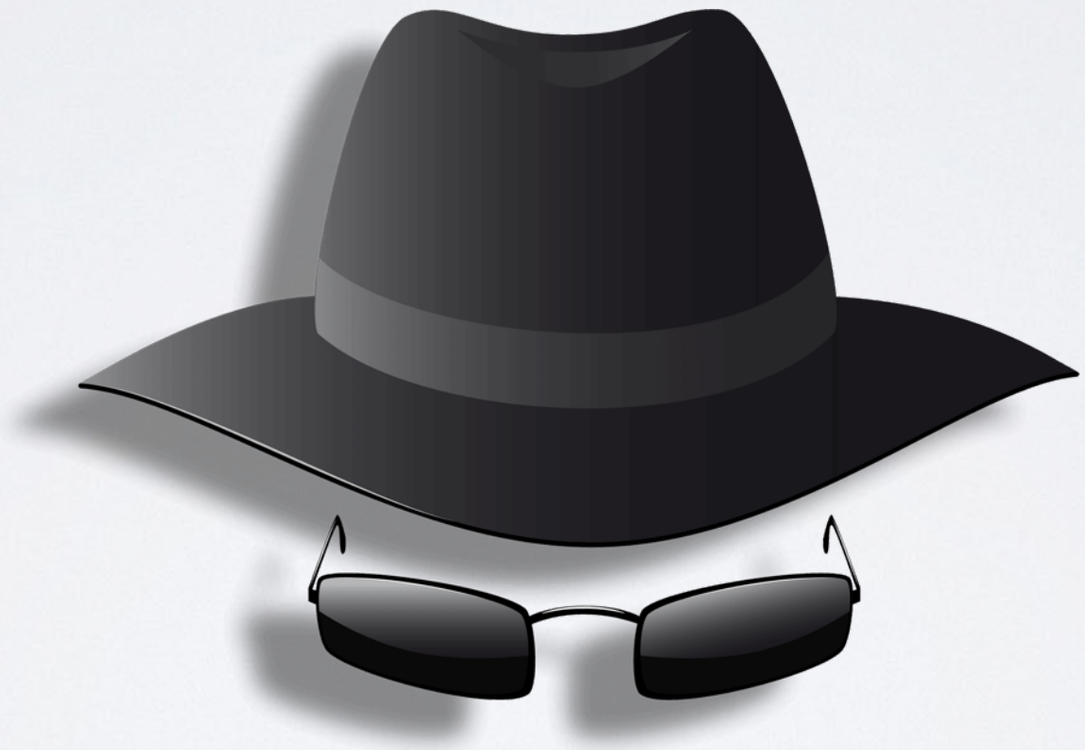
familiar environment

driver ecosystem





protection/isolation



OUTLINE

Background/Overview

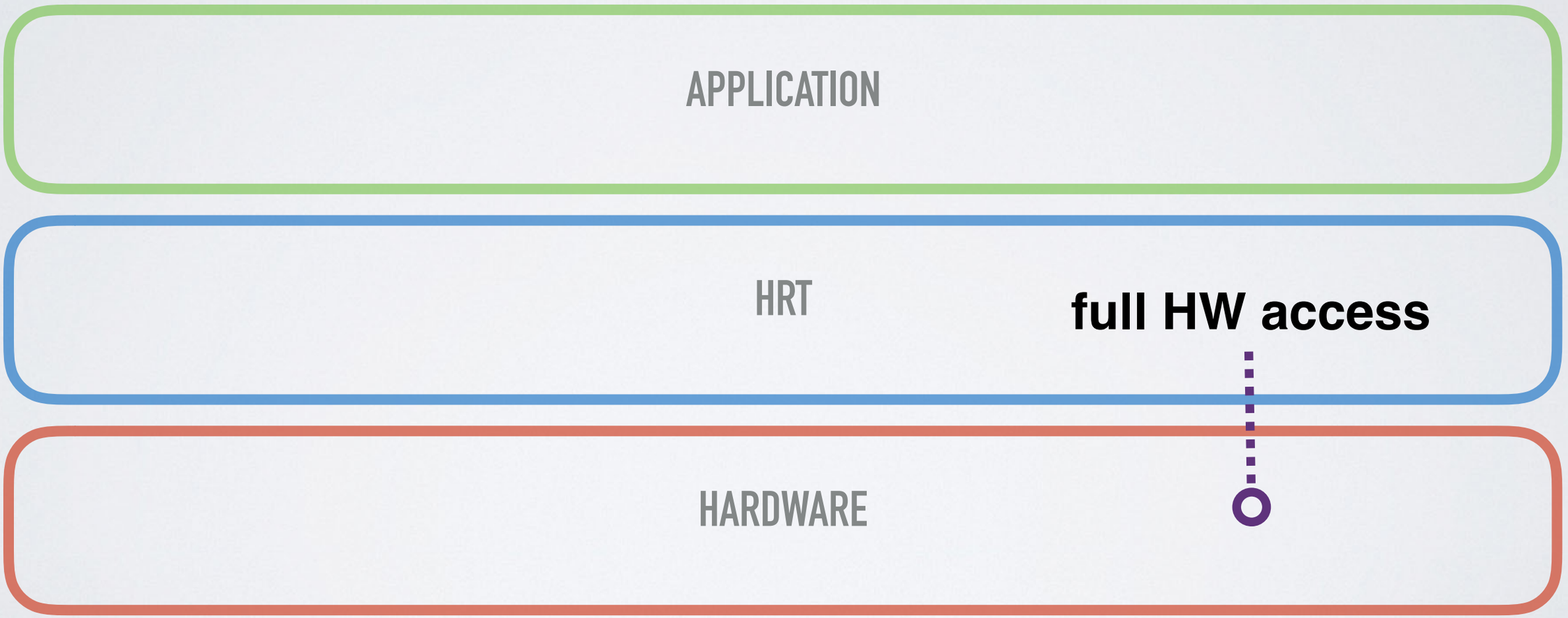
Nautilus

Deployment Models

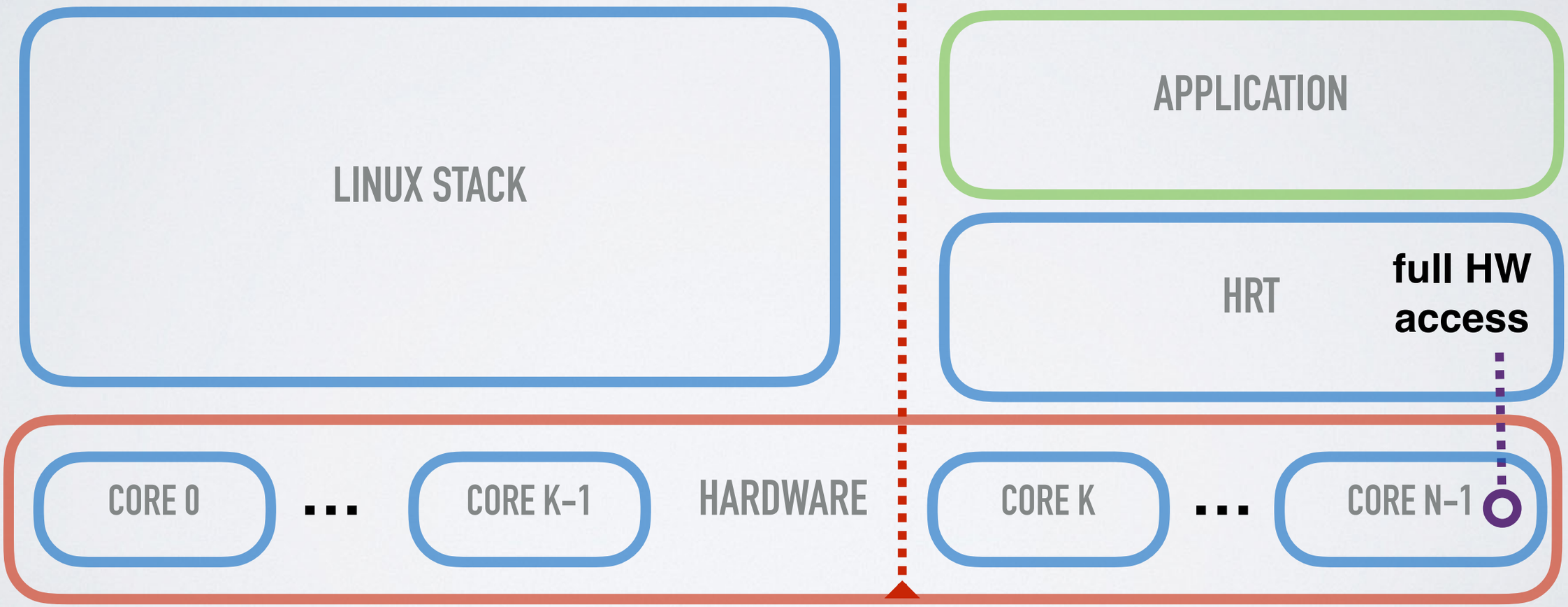
Hybrid Virtual Machine

Multiverse & Future Work

DEDICATED



PARTITIONED



VM

LINUX STACK

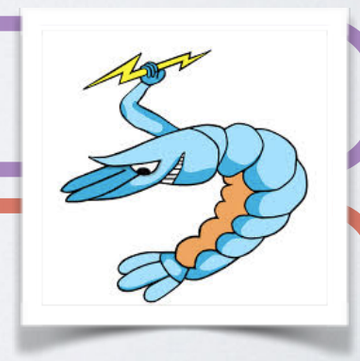
PARALLEL APP

HRT

VIRTUAL MACHINE 0

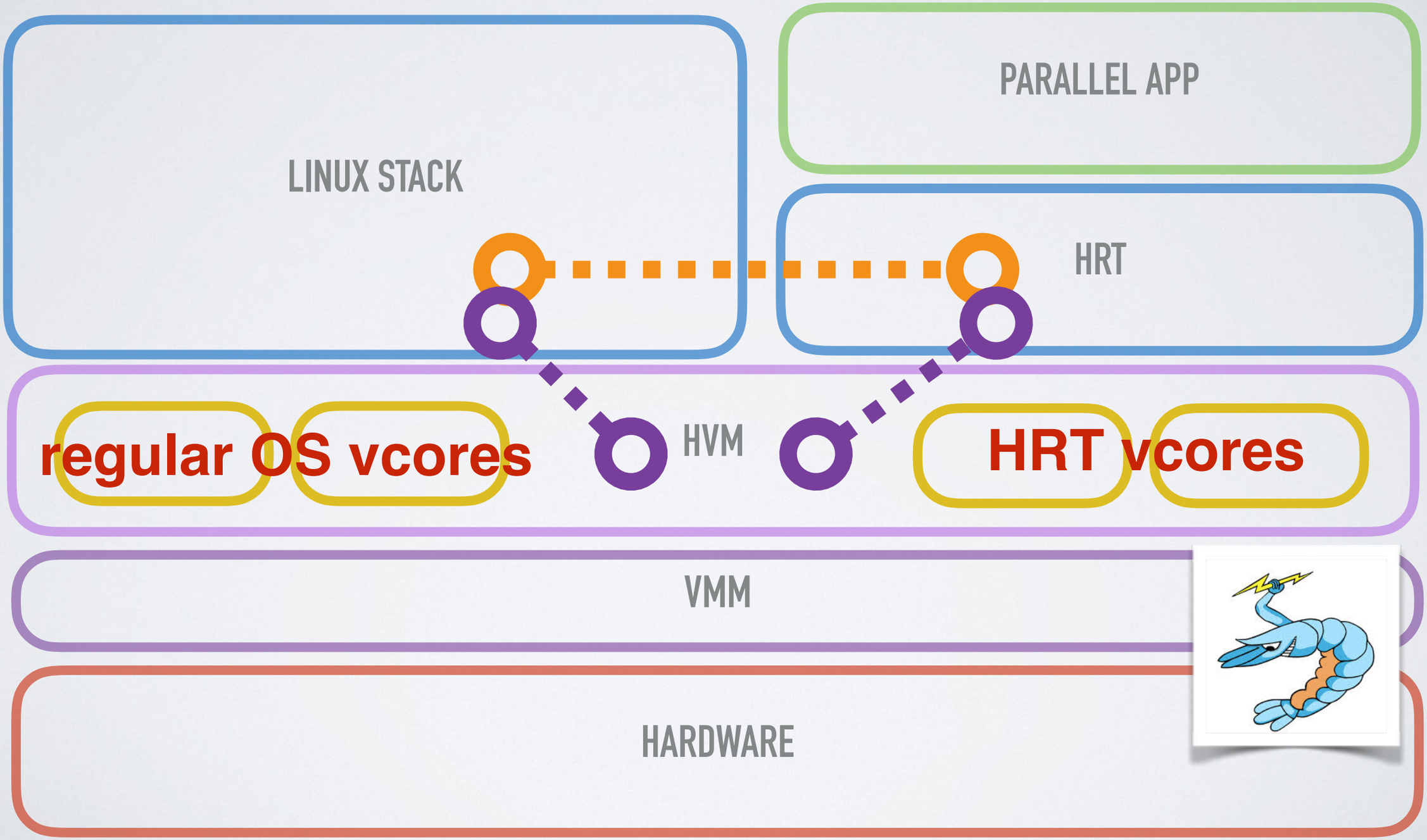
VIRTUAL MACHINE 1

VMM



HARDWARE

Hybrid Virtual Machine



OUTLINE

Background/Overview

Nautilus

Deployment Models

Hybrid Virtual Machine

Multiverse & Future Work

regular OS (ROS)

legacy functionality from ROS via HVM

performance path

PARALLEL APP

PARALLEL RUNTIME

user-mode

kernel-mode

GENERAL PURPOSE OS

user-mode

kernel-mode

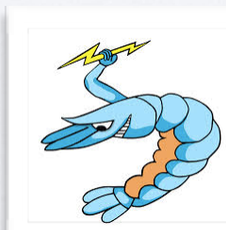
PARALLEL APP

HRT

NODE HARDWARE

GENERAL VIRTUALIZATION MODEL

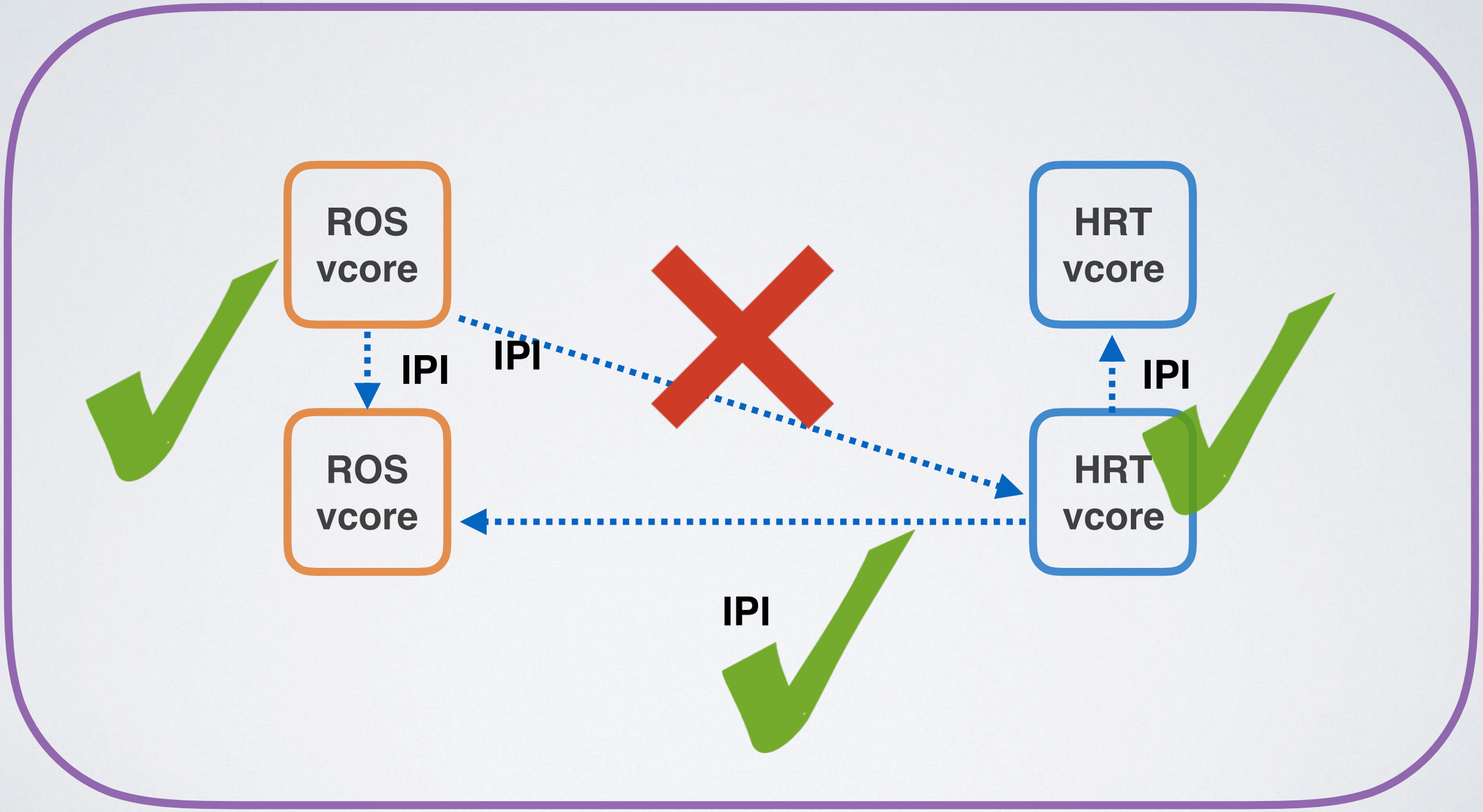
HVM



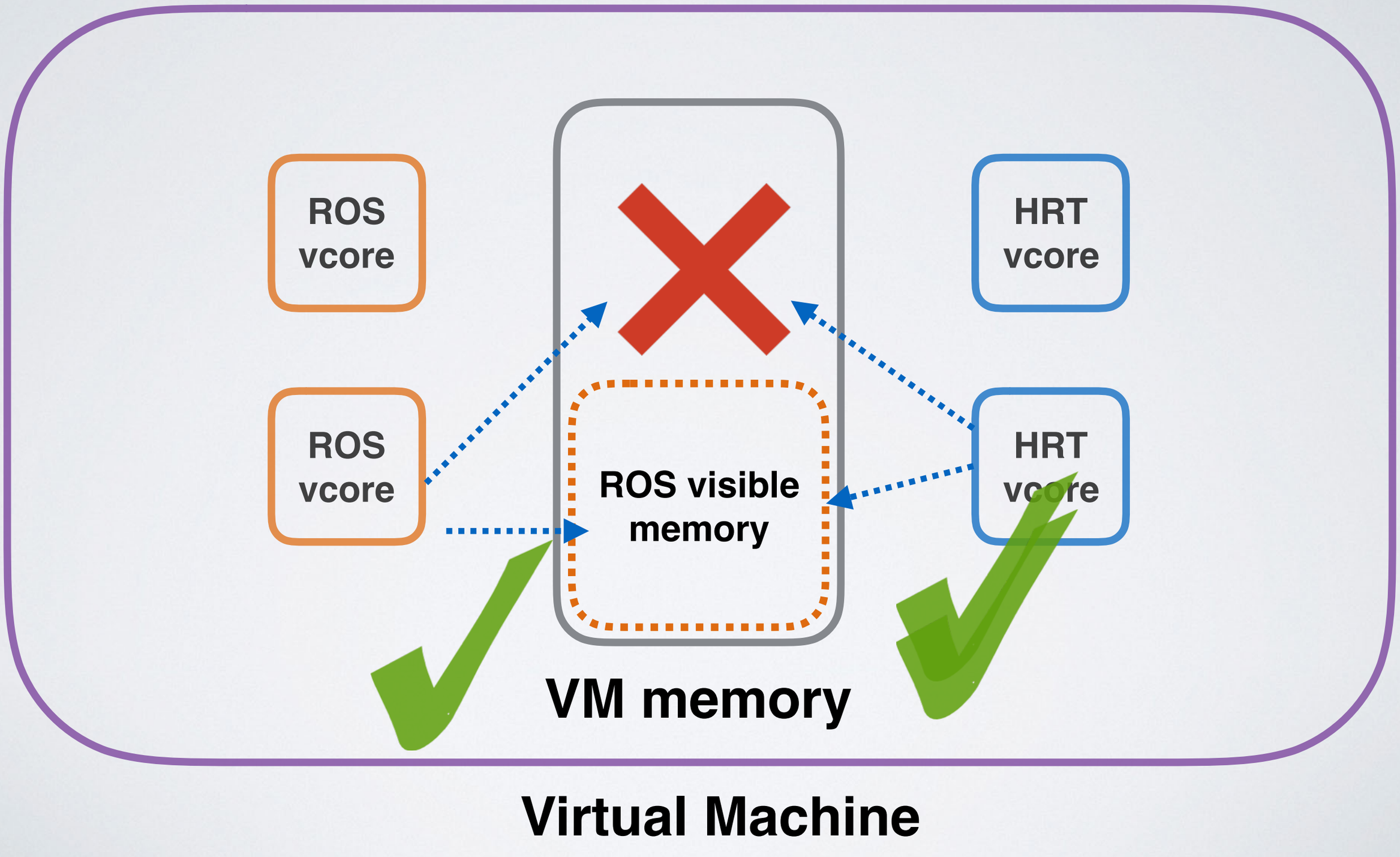
SPECIALIZED VIRTUALIZATION MODEL

NODE HARDWARE

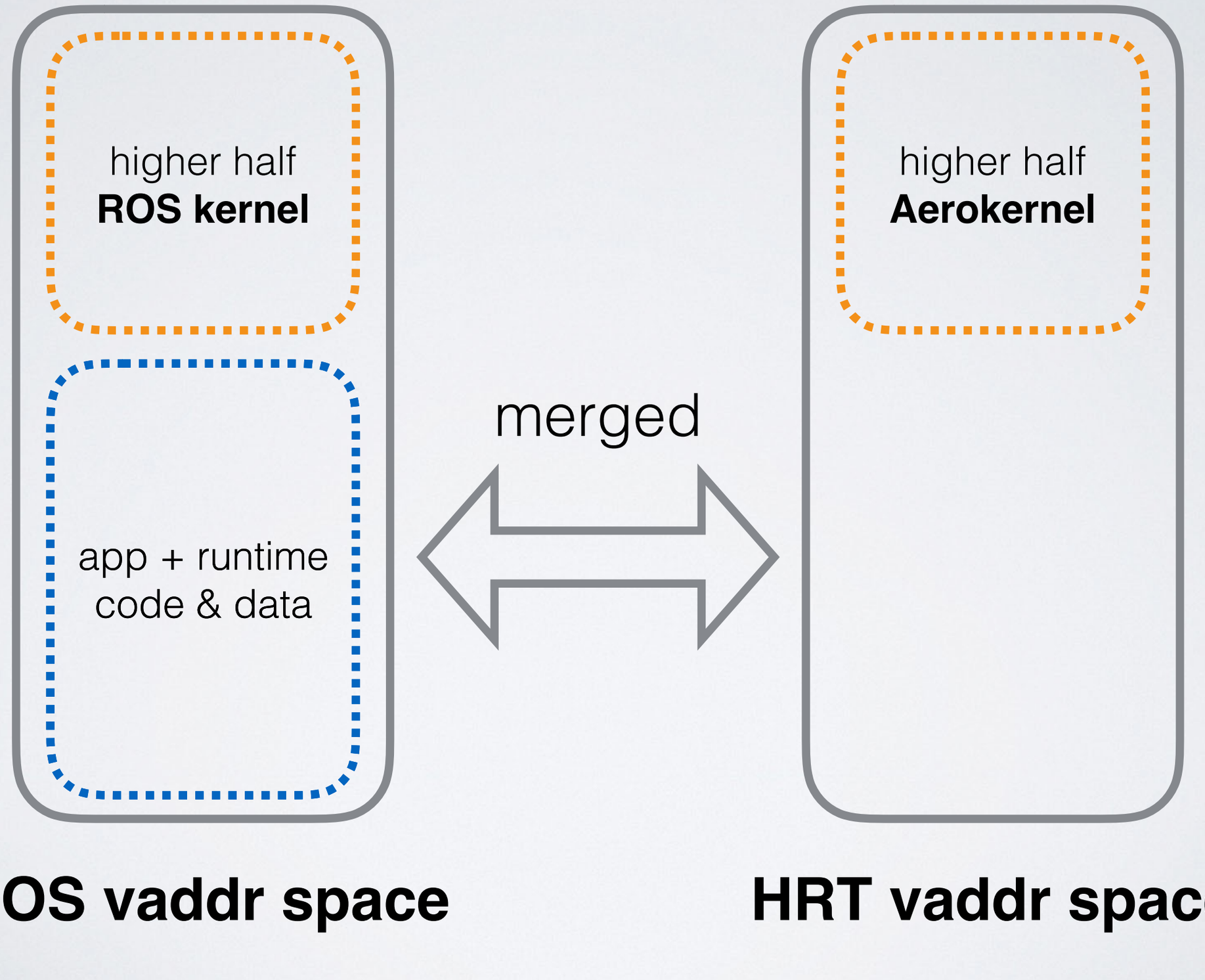
ROS/HRT setup



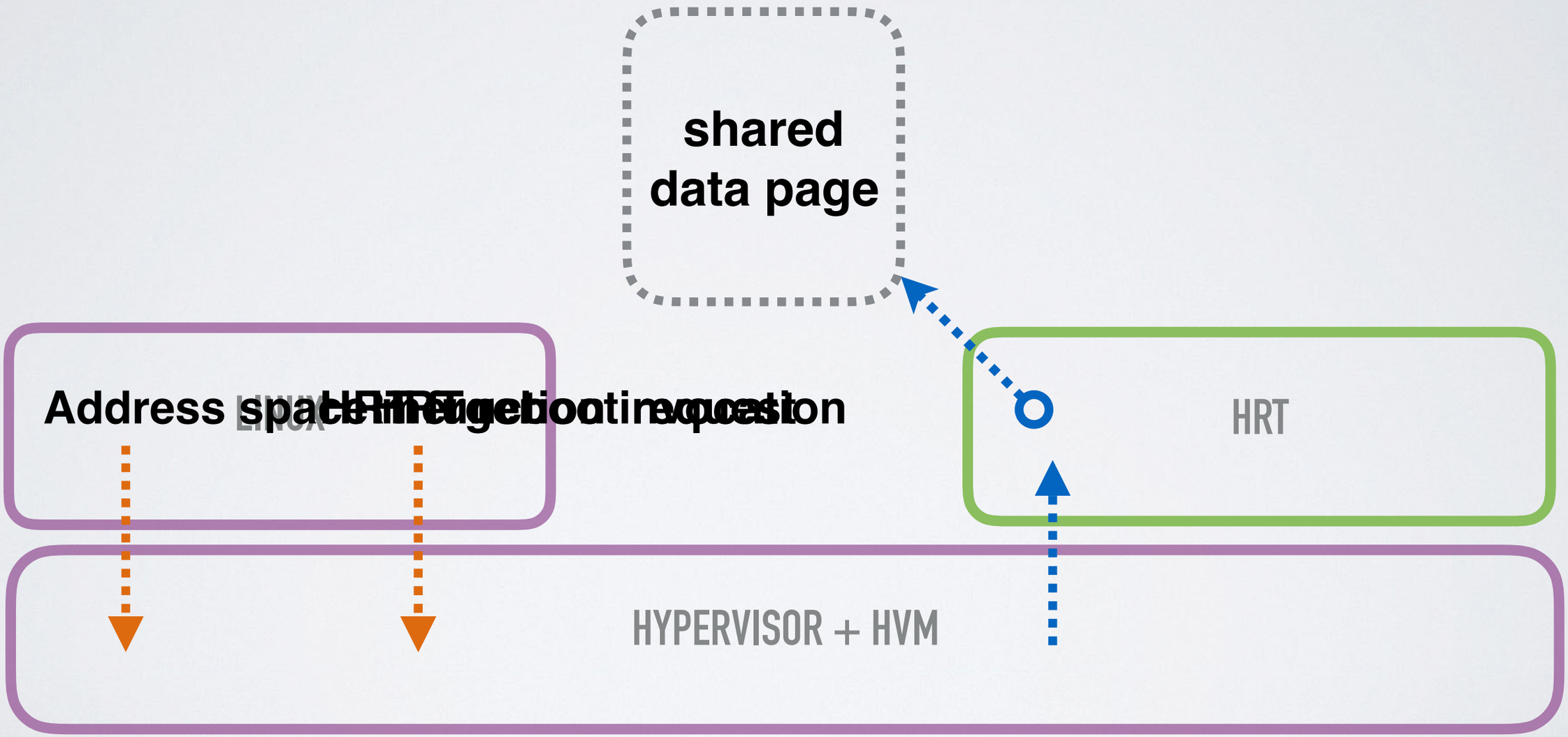
ROS/HRT setup



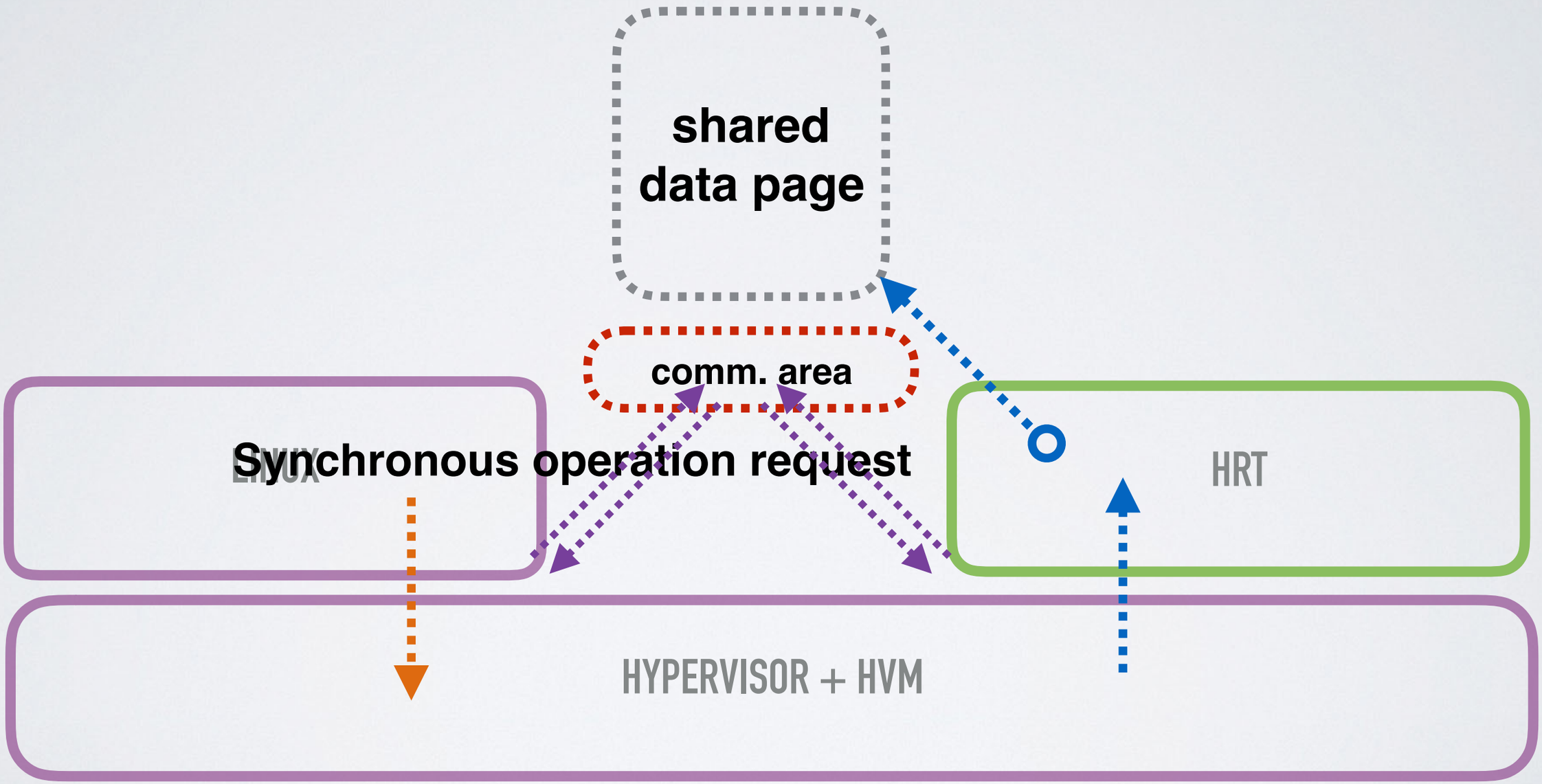
merged address space



ROS/HRT communication



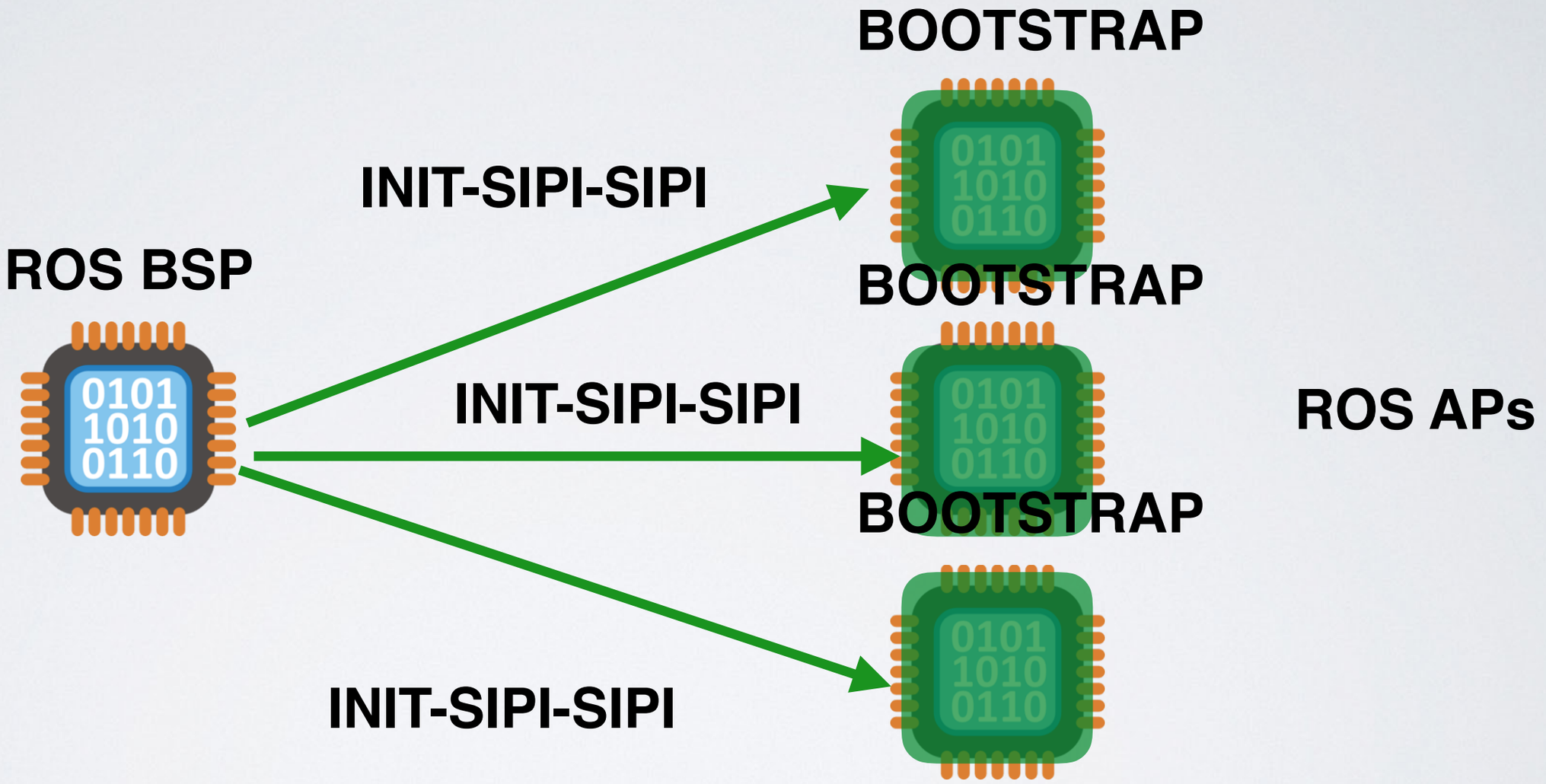
ROS/HRT communication



ROS/HRT communication

	Cycles	Time
Addr space merge	~33K	15μs
Asynch function invocation	~25K	11μs
Synch function invocation (remote socket)	~1060	482ns
Synch function invocation (same socket)	~790	359ns

ROS boot

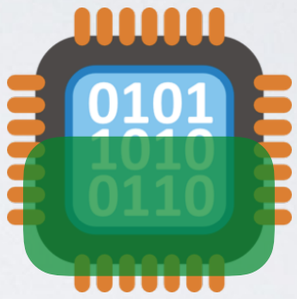

BOOTSTRAP
INIT-SIPI-SIPI
ROS BSP
BOOTSTRAP
INIT-SIPI-SIPI
ROS APs
BOOTSTRAP
INIT-SIPI-SIPI

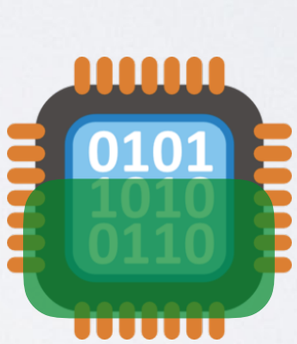
HRT boot

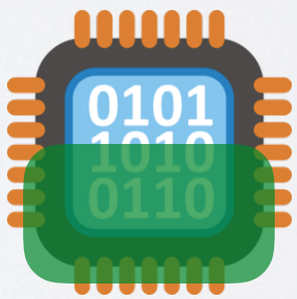
HVM bootstrap

registers,in

system







GDT

IDT

TSS

REGISTERS

INIT
PAGE
TABLES

*no need for INIT-SIPI-SIPI sequence

LINUX FORK + EXEC ~ 714µs

HVM + HRT CORE BOOT ~ *61µs*

how do we take a legacy runtime to the HRT + X model?

PORT

porting a runtime/app to an a new OS environment is...

DIFFICULT

TIME-CONSUMING

ERROR-PRONE

development cycle:

do {

ADD FUNCTION

REBUILD

BOOT HRT

} while (HRT falls over)

much of the functionality is

NOT ON THE CRITICAL PATH

we want to make this

easier

OUTLINE

Background/Overview

Nautilus

Deployment Models

Hybrid Virtual Machine

Multiverse & Future Work

give us your legacy

runtime

we automatically

transform it to run

as an HRT

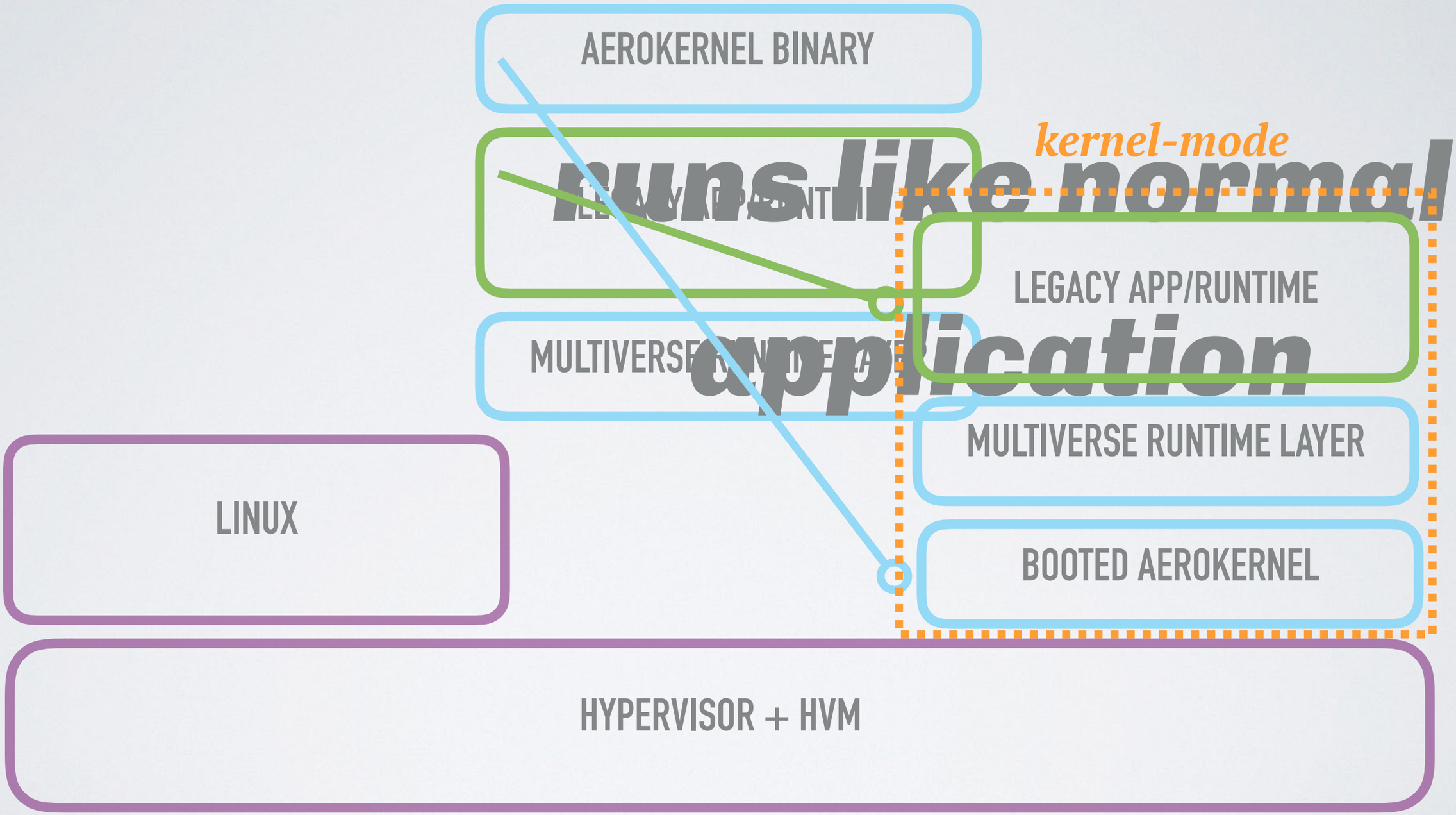
in kernel mode

bridged with a legacy OS

(AUTOMATIC HYBRIDIZATION)

LEGACY APP/RUNTIME

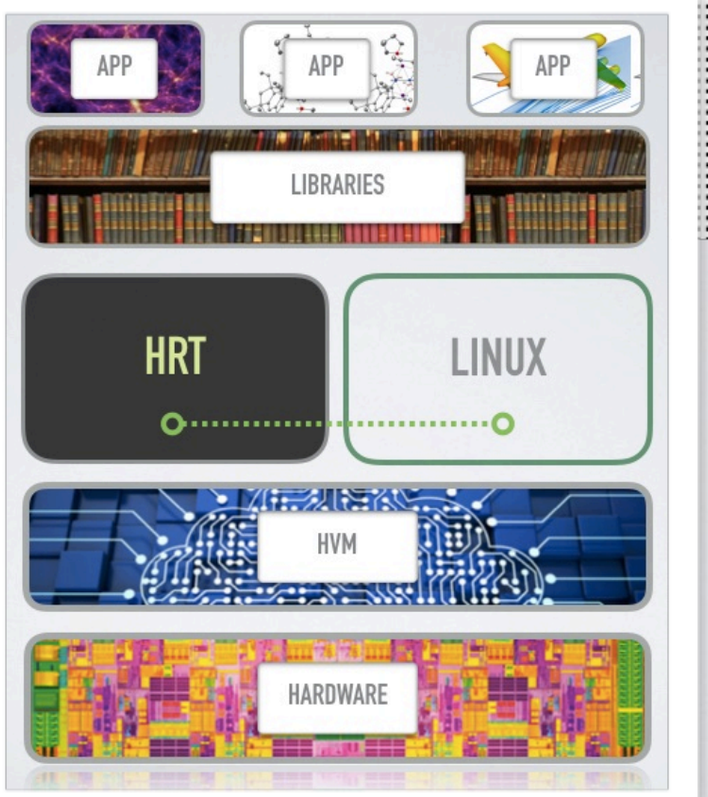
rebuild with our toolchain


AEROKERNEL BINARY
kernel-mode
runs like normal!
LEGACY APP/RUNTIME
application
MULTIVERSE RUNTIME LAYER
BOOTED AEROKERNEL
LINUX
HYPERVISOR + HVM

```
#
# ls
bench-write.out      go                   mracket-GOLD
binary-tree-2.rkt    intsum-native        multiverse-racket
bytes                ism                  multiverse.log
collects             isn                  nbody.rkt
doall.sh             lgn-hpcg             racket
doruns.sh            lgo                  results
fannkuch-redux.rkt   lost+found           spectral-norm.rkt
fasta-3.rkt          lpm                  test.out
fasta.rkt            lpn                  test.t
g                    mandelbrot-2.rkt
#
```

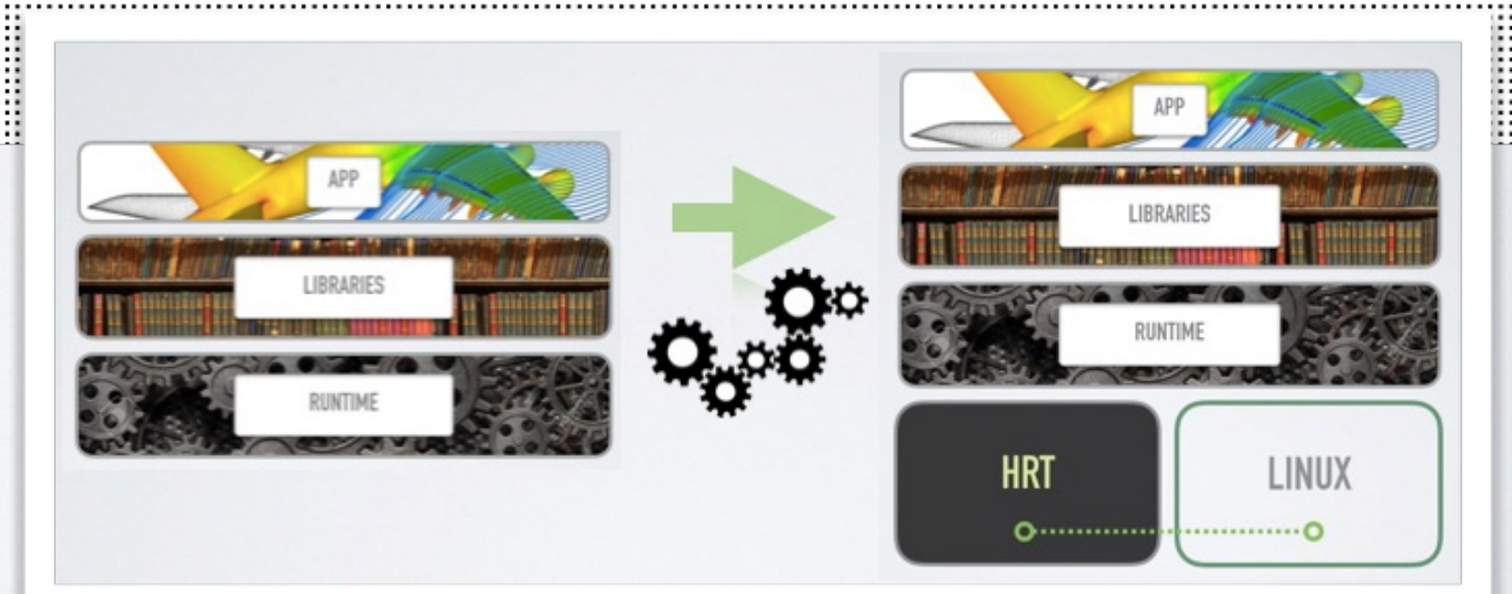
4 parallel runtimes (Legion, Racket, NDPC, NESL)

nautilus



Hybrid Virtual Machine

bridge HRT with legacy OS



Multiverse

automatic transformation: legacy app+runtime → HRT

thank you

my webpage: halek.co
lab: presciencelab.org
download nautilus: nautilus.halek.co
v3vee project: v3vee.org

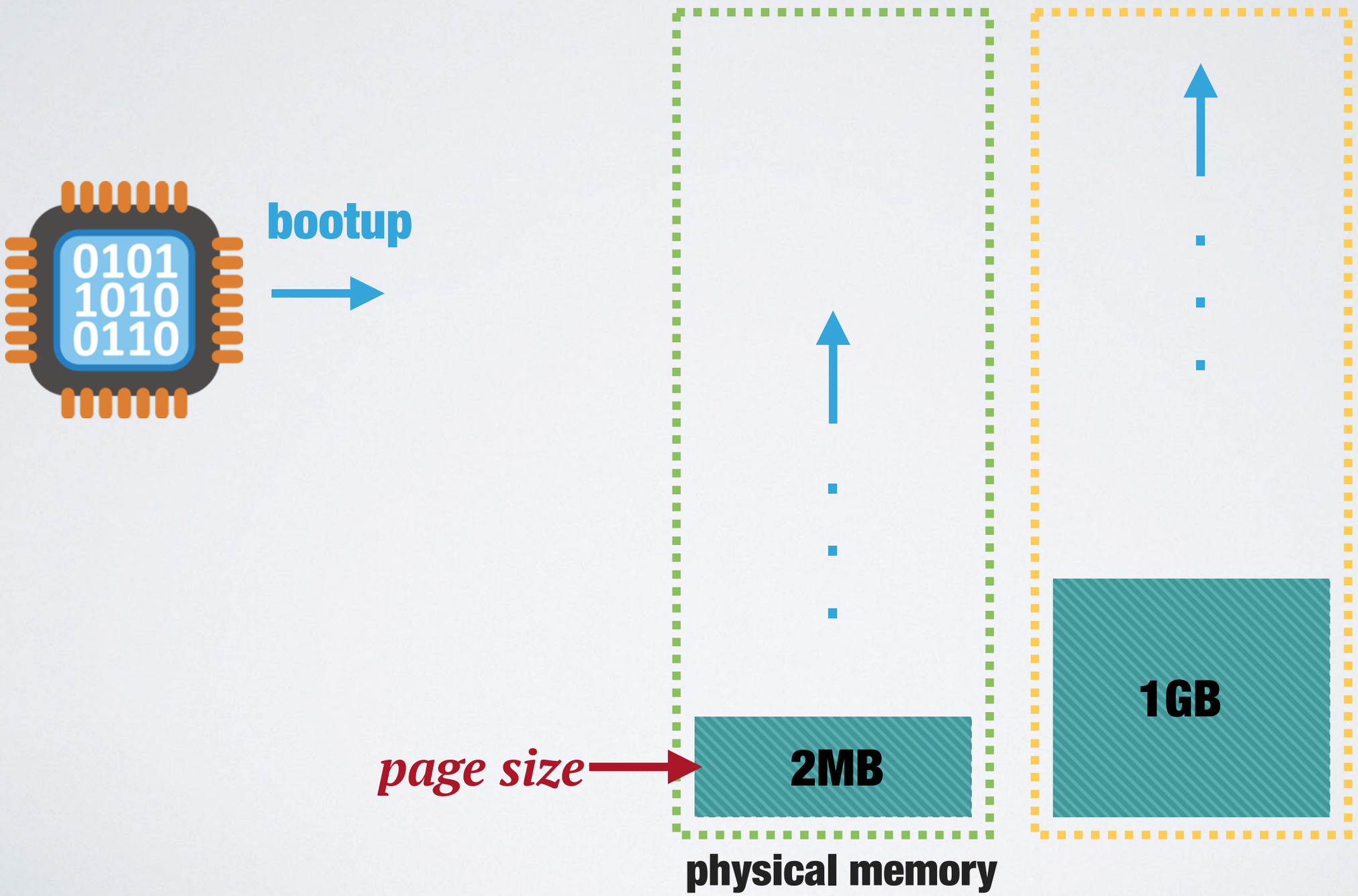
backups

thread fork

interrupt driven execution

memory and paging

static identity map



eliminates expensive page faults

reduces TLB misses + shootdowns

increases performance under virtualization

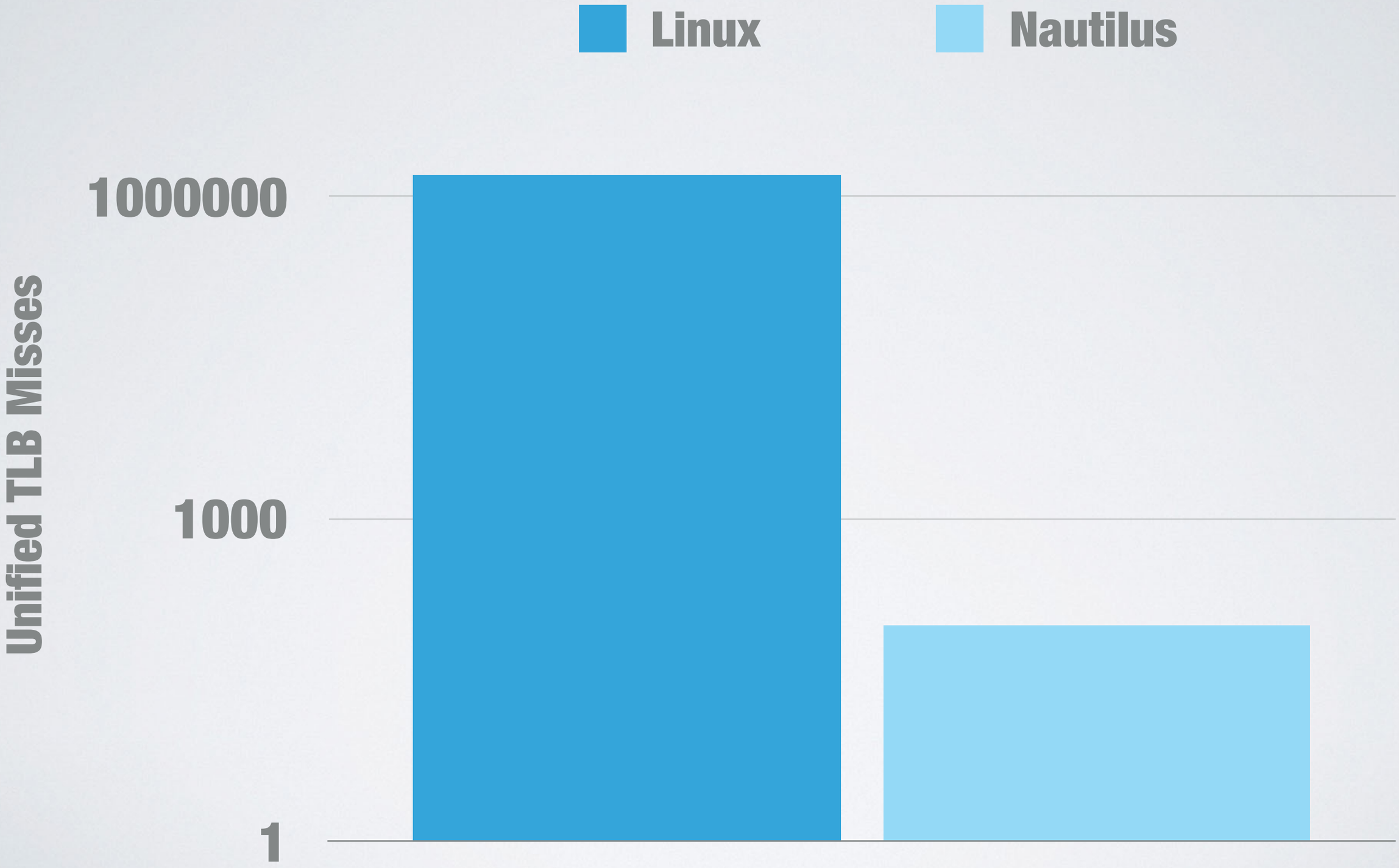
log scale

1000000

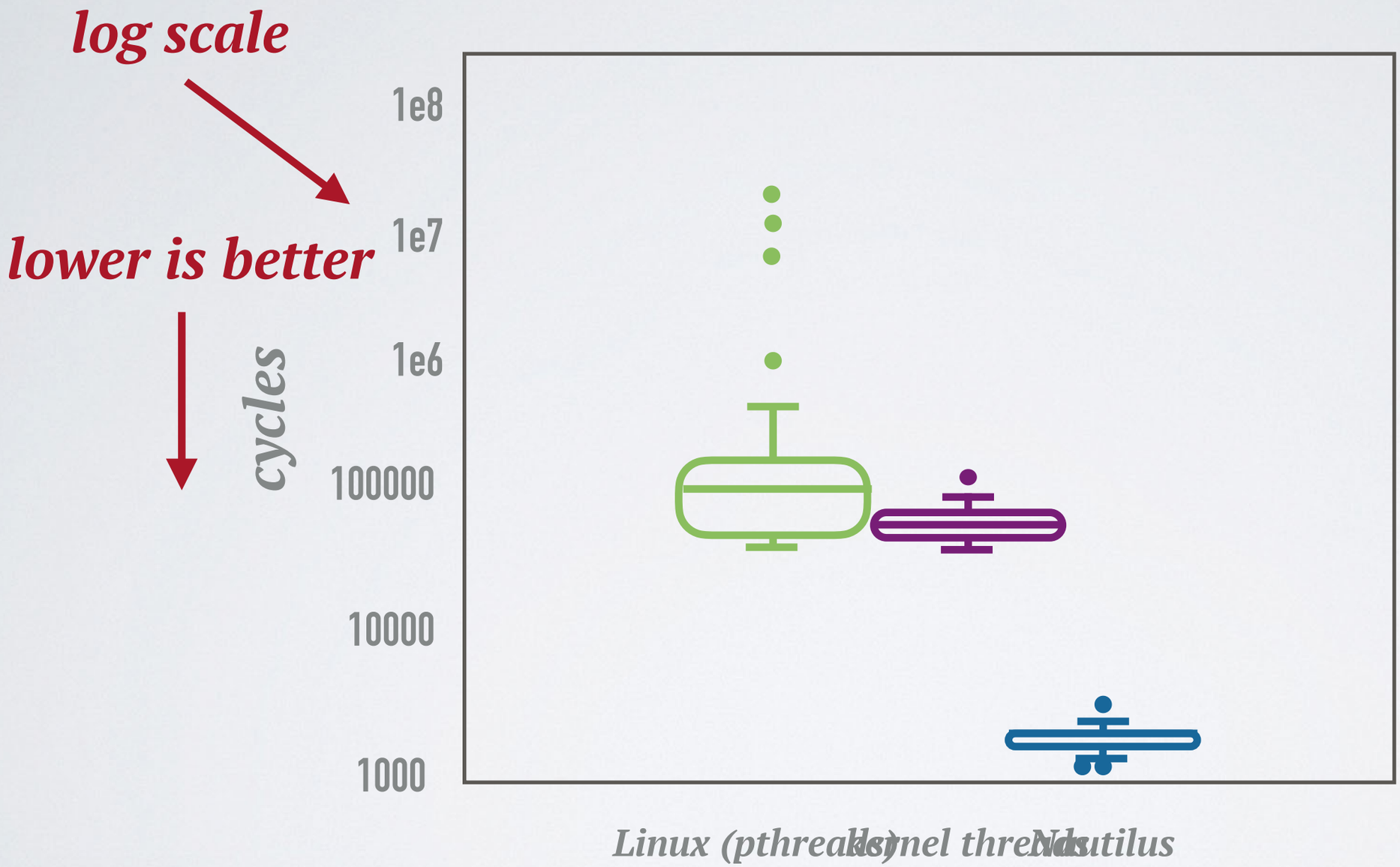
Unified TLB Misses

1000

1


Linux
Nautilus
1000000
1000
1
Unified TLB Misses

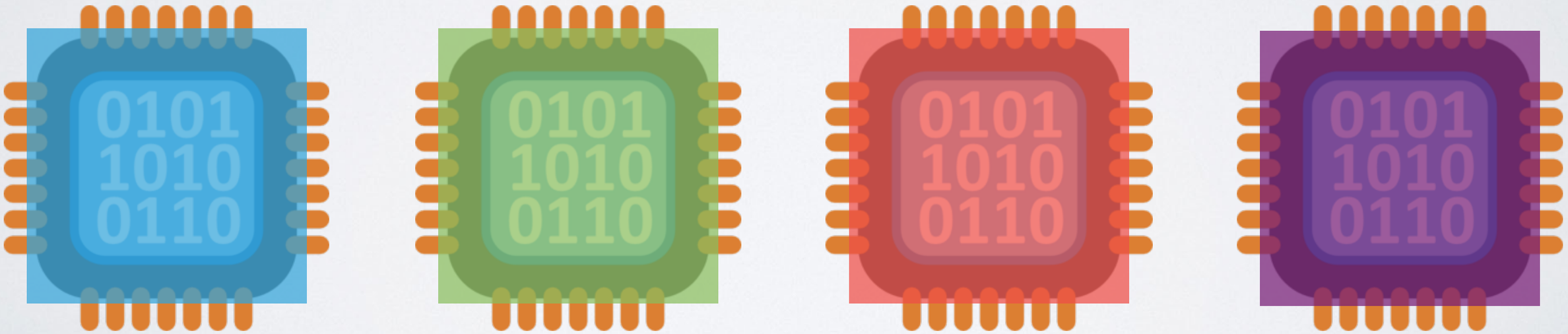
thread creation is **PREDICTABLE**



x86_64 Opteron: 64 cores, 4 sockets, 8 numa zones, 128GB RAM

binding to physical processor is

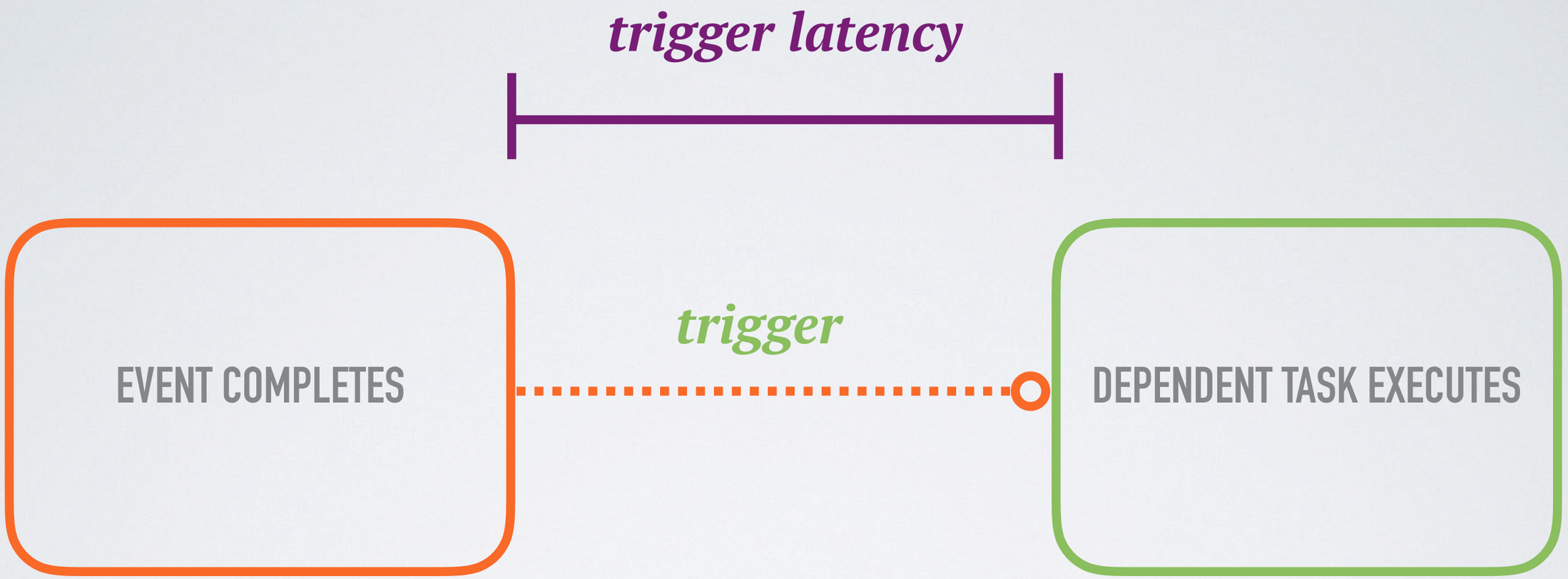
GUARANTEED



RUNTIME *control over physical CPU*

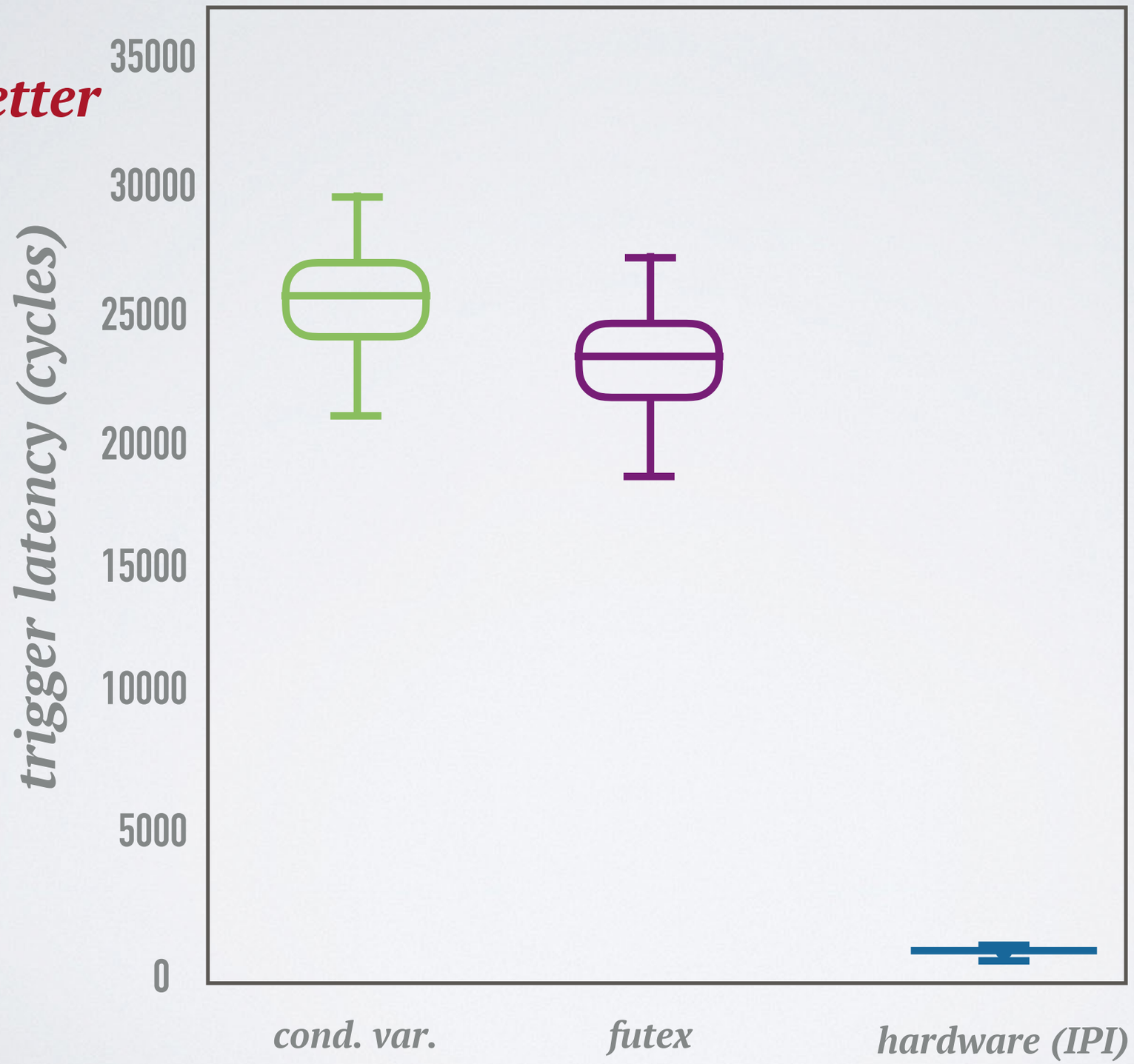
interrupts & events

asynchronous notifications


trigger latency
trigger
EVENT COMPLETES
DEPENDENT TASK EXECUTES

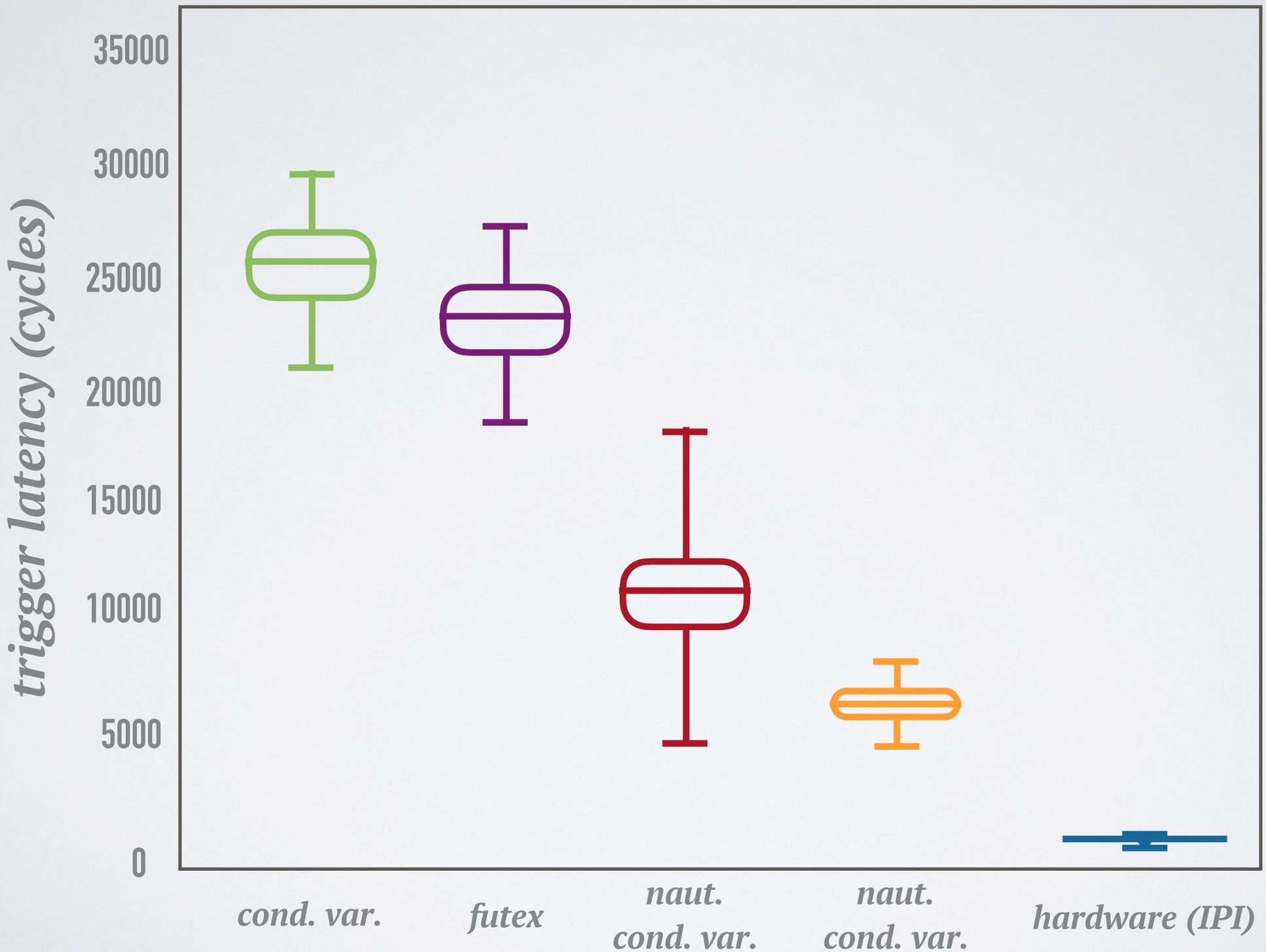
user-mode software events are **SLOW**

lower is better



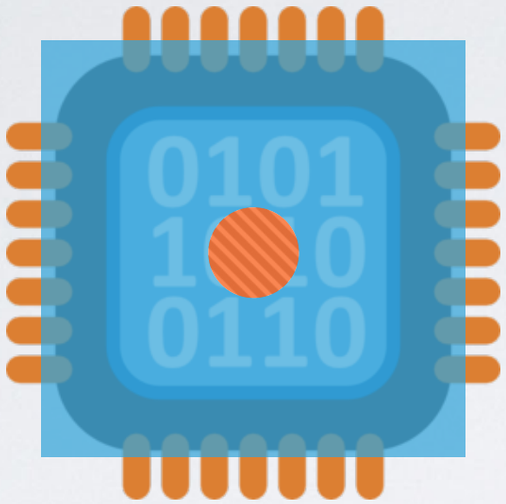
x86_64 Opteron: 64 cores, 4 sockets, 8 numa zones, 128GB RAM

nautilus events triggers are **FASTER**

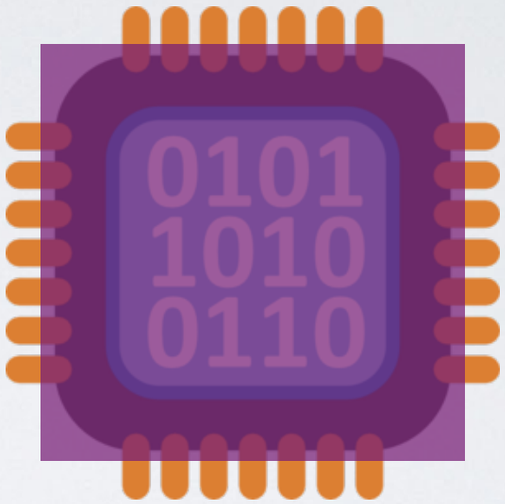


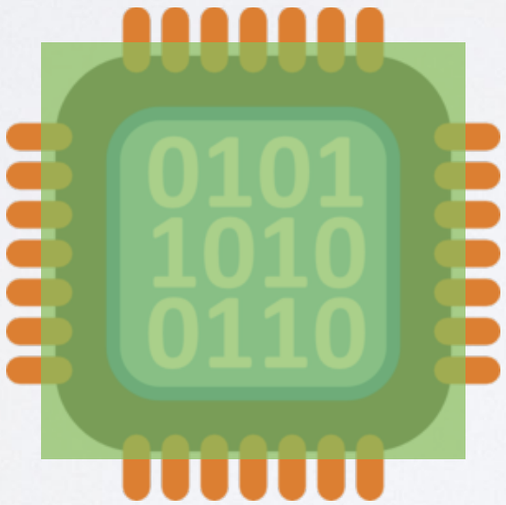
x86_64 Opteron: 64 cores, 4 sockets, 8 numa zones, 128GB RAM

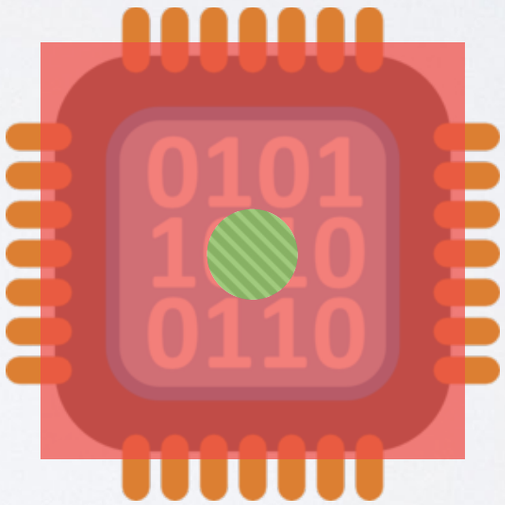
handle_event()

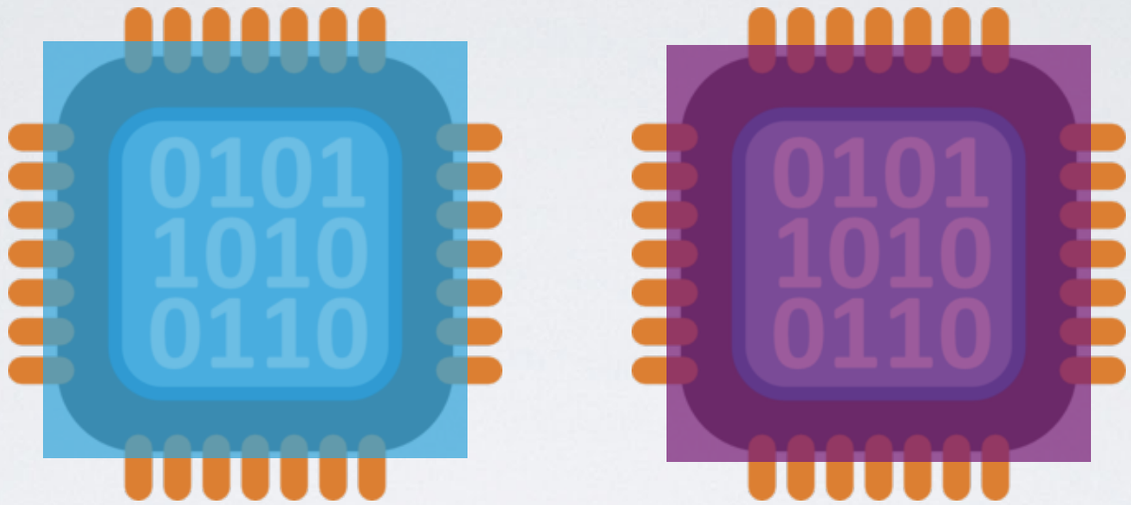


handle_event()

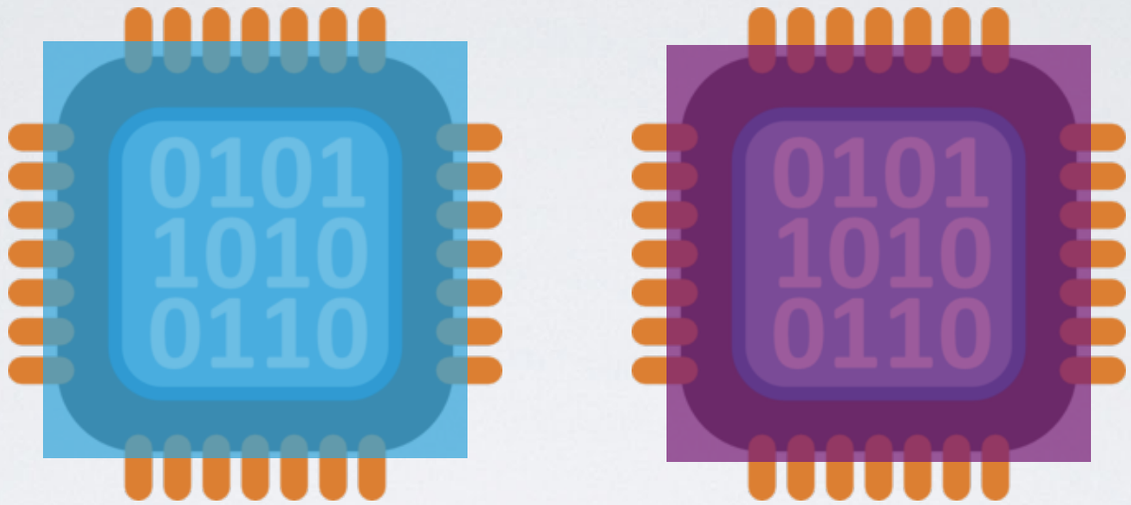


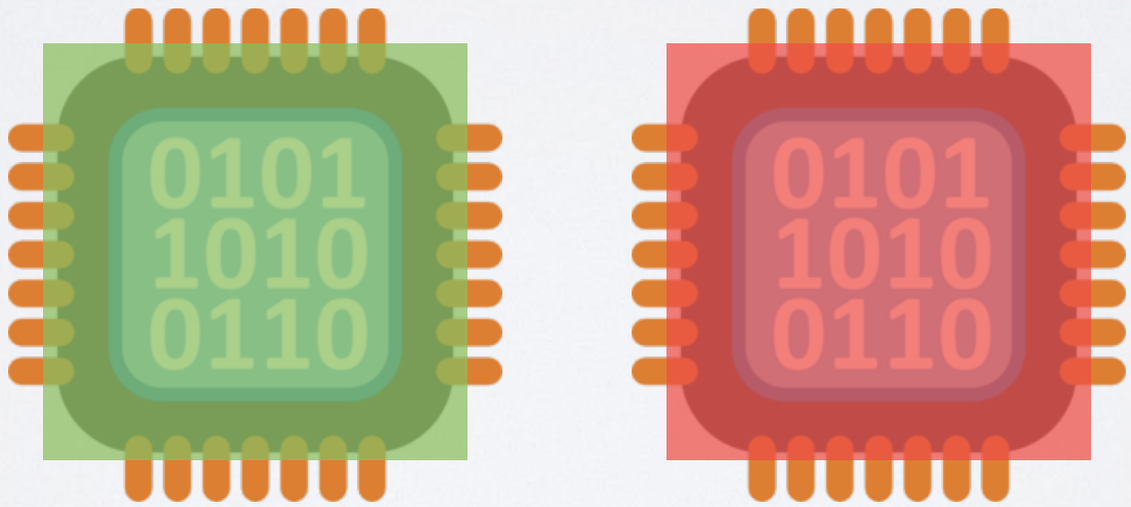






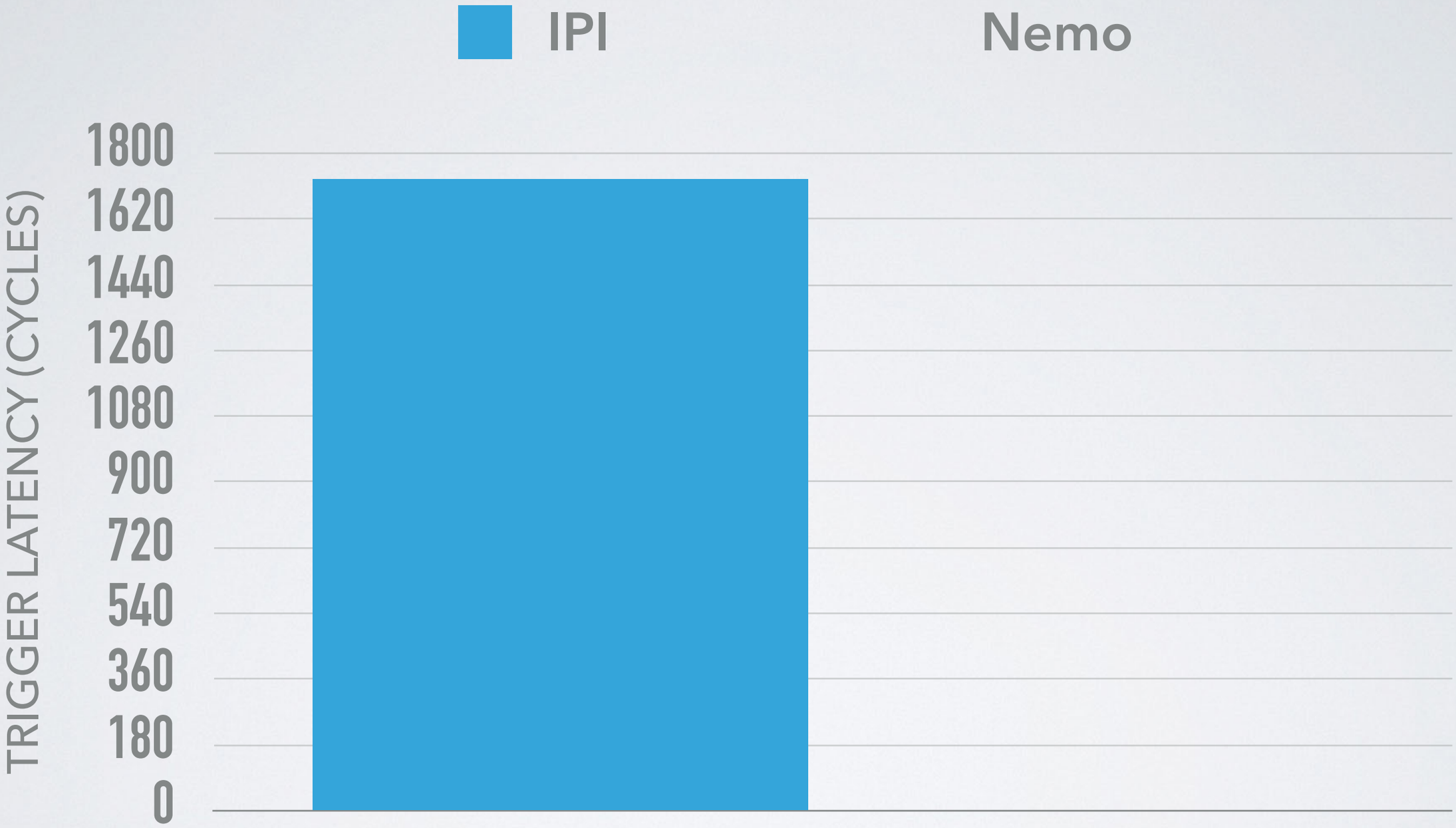
asynchronous, IPI-based execution

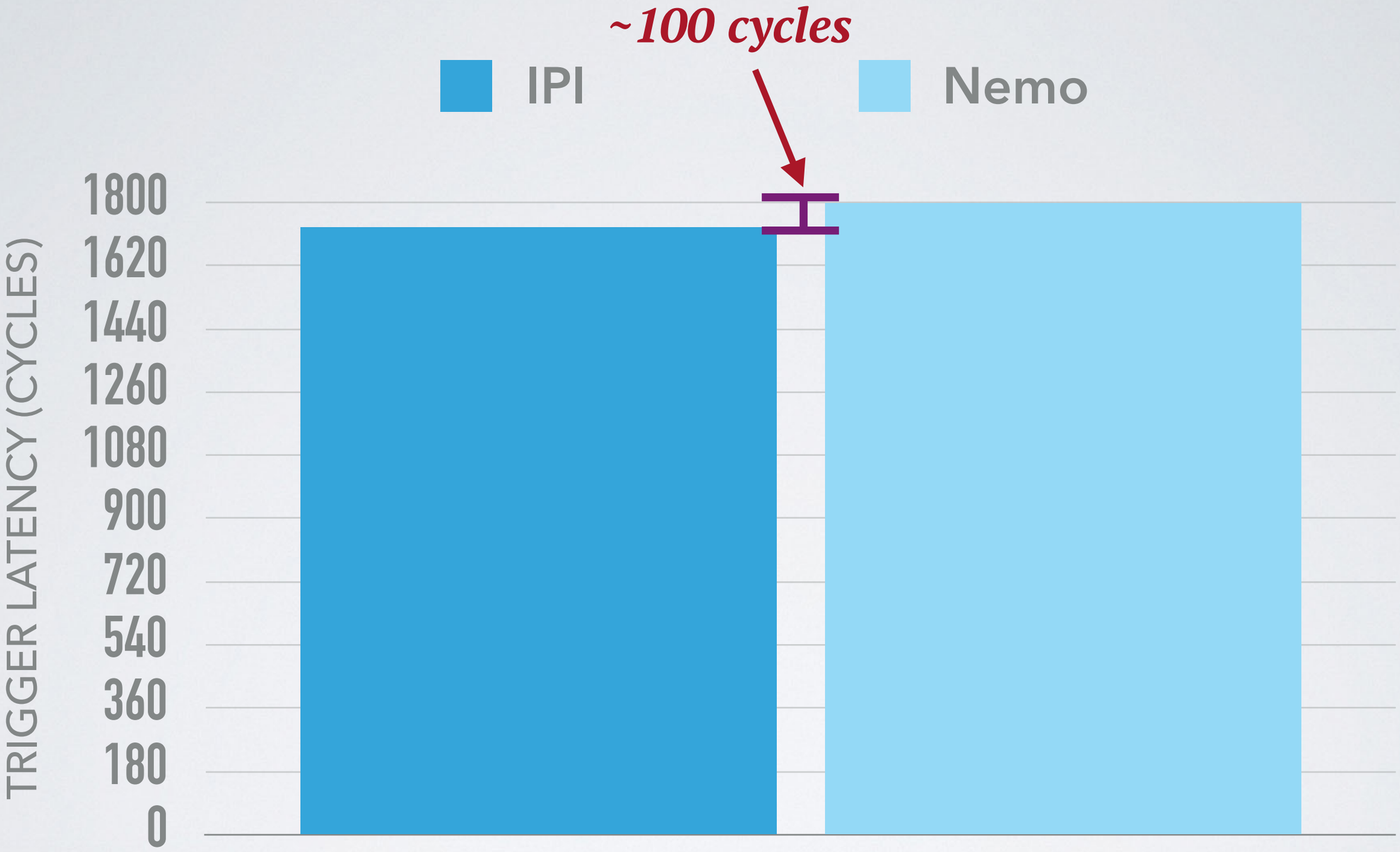




NOT POSSIBLE IN LINUX USERSPACE

lower is better


IPI
Nemo
TRIGGER LATENCY (CYCLES)
1800
1620
1440
1260
1080
900
720
540
360
180
0


~100 cycles
IPI
Nemo
TRIGGER LATENCY (CYCLES)
1800
1620
1440
1260
1080
900
720
540
360
180
0

this gives us an

incremental path

for creating HRTs

start out with a working HRT system

then pull functionality into the HRT for hot spots

RACKET

most widely used Scheme implementation

downloaded ~300 times/day

complex runtime with JIT

ROS/HRT communication

AEROKERNEL BINARY

LEGACY APP/RUNTIME

MULTIVERSE RUNTIME LAYER

LINUX

system call

HYPERVISOR + HVM

system call

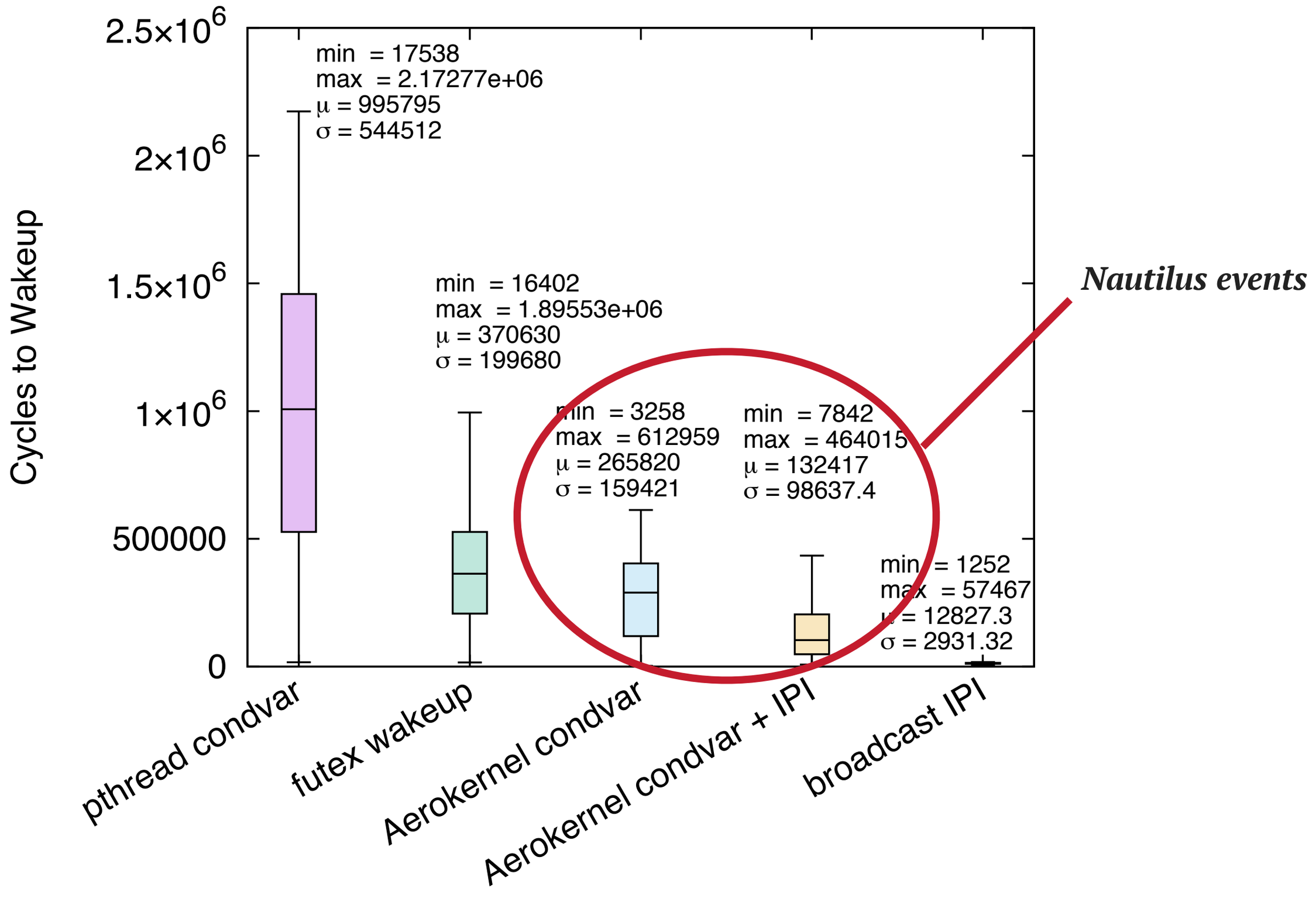
LEGACY APP/RUNTIME

MULTIVERSE RUNTIME LAYER

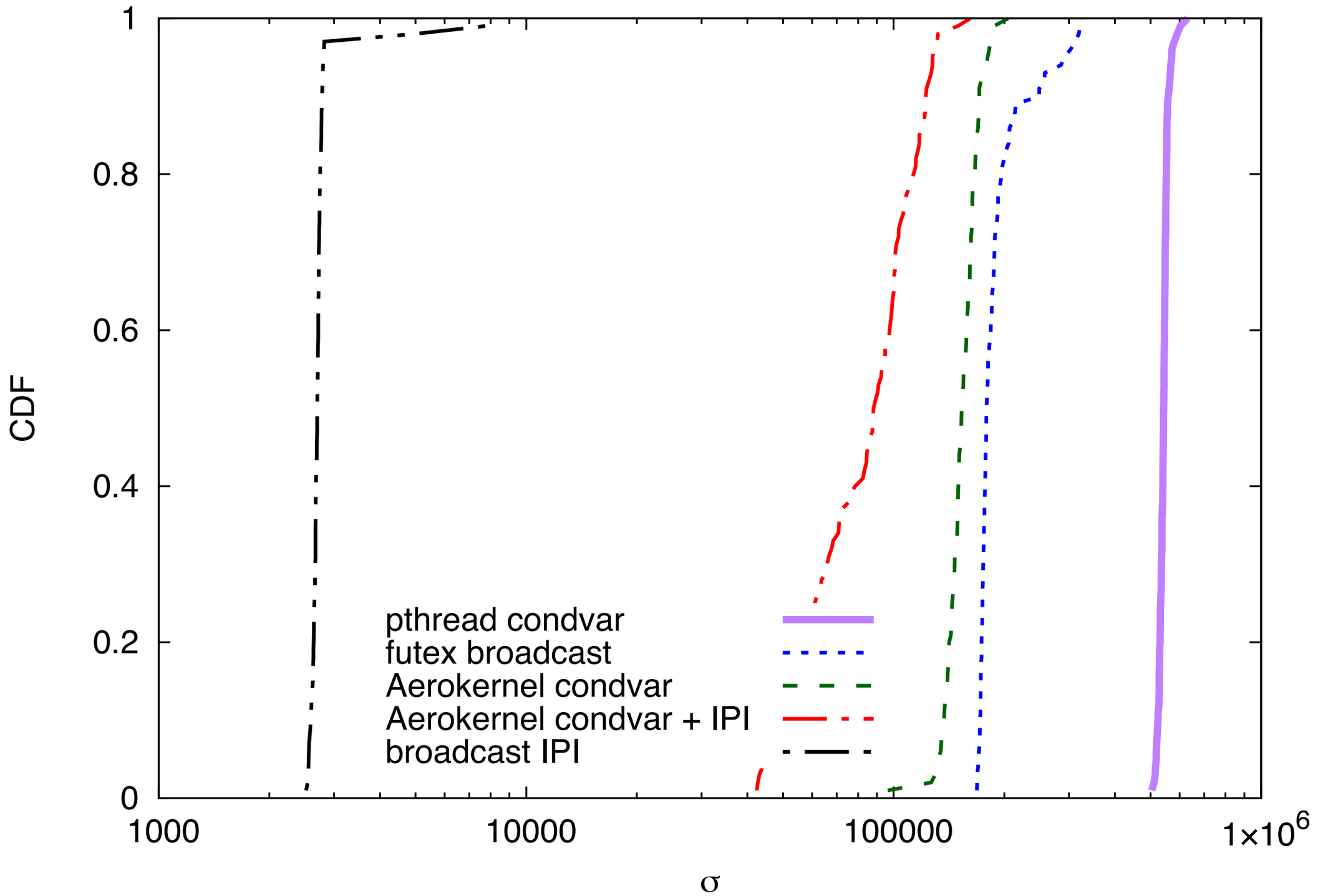
BOOTED AEROKERNEL

```
# ls
bench-write.out      go                   mracket-GOLD
binary-tree-2.rkt    intsum-native        multiverse.log
bytes                ism                  nbody.rkt
collects             isn                  racket
doall.sh             lgn-hpcg             results
doruns.sh            lgo                  spectral-norm.rkt
fannkuch-redux.rkt   lost+found           test.out
fasta-3.rkt          lpm                  test.t
fasta.rkt            lpn
g                    mandelbrot-2.rkt
#
```

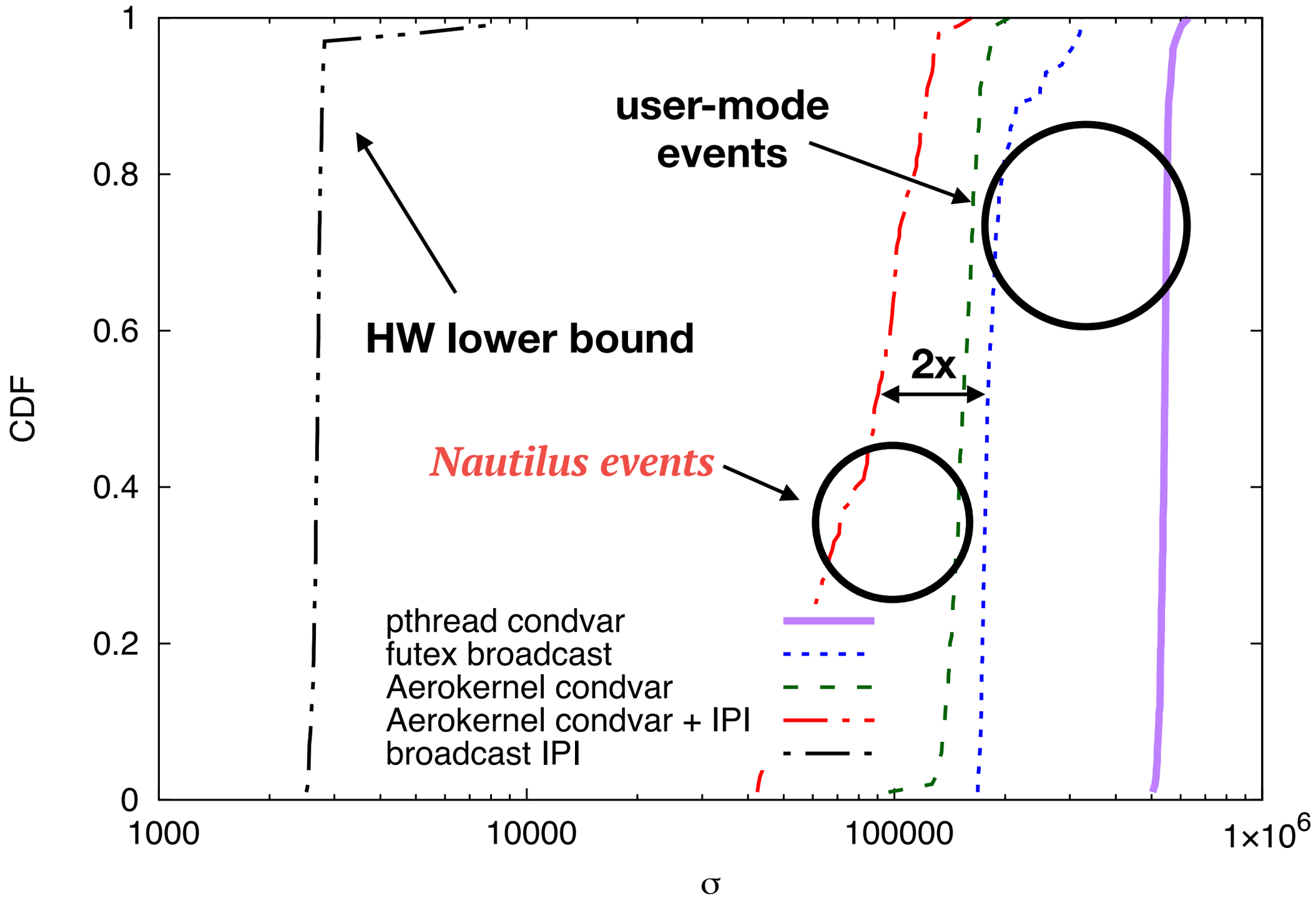
Broadcast wakeups on x64



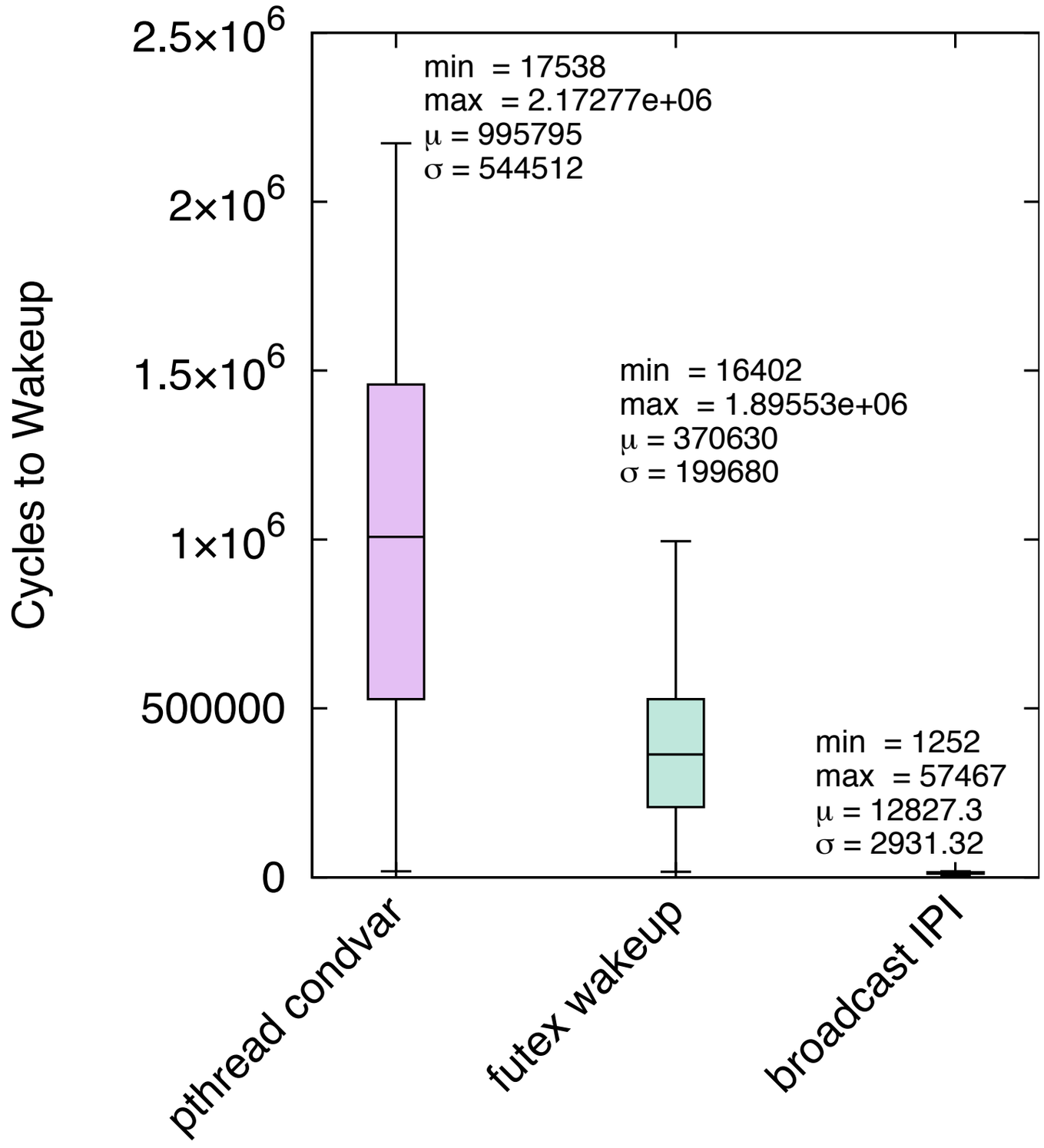
Wakeup deviation on x64



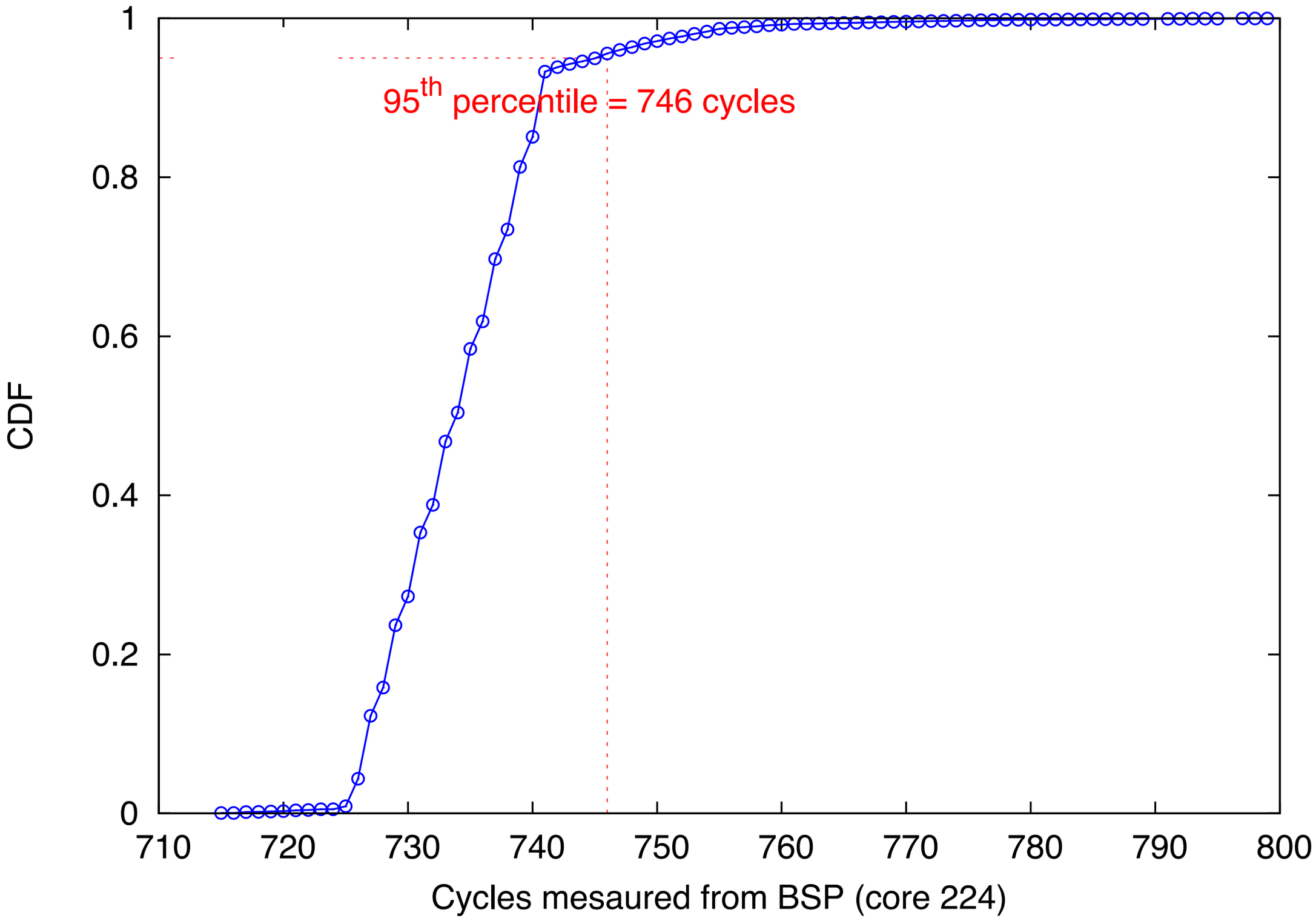
Wakeup deviation on x64



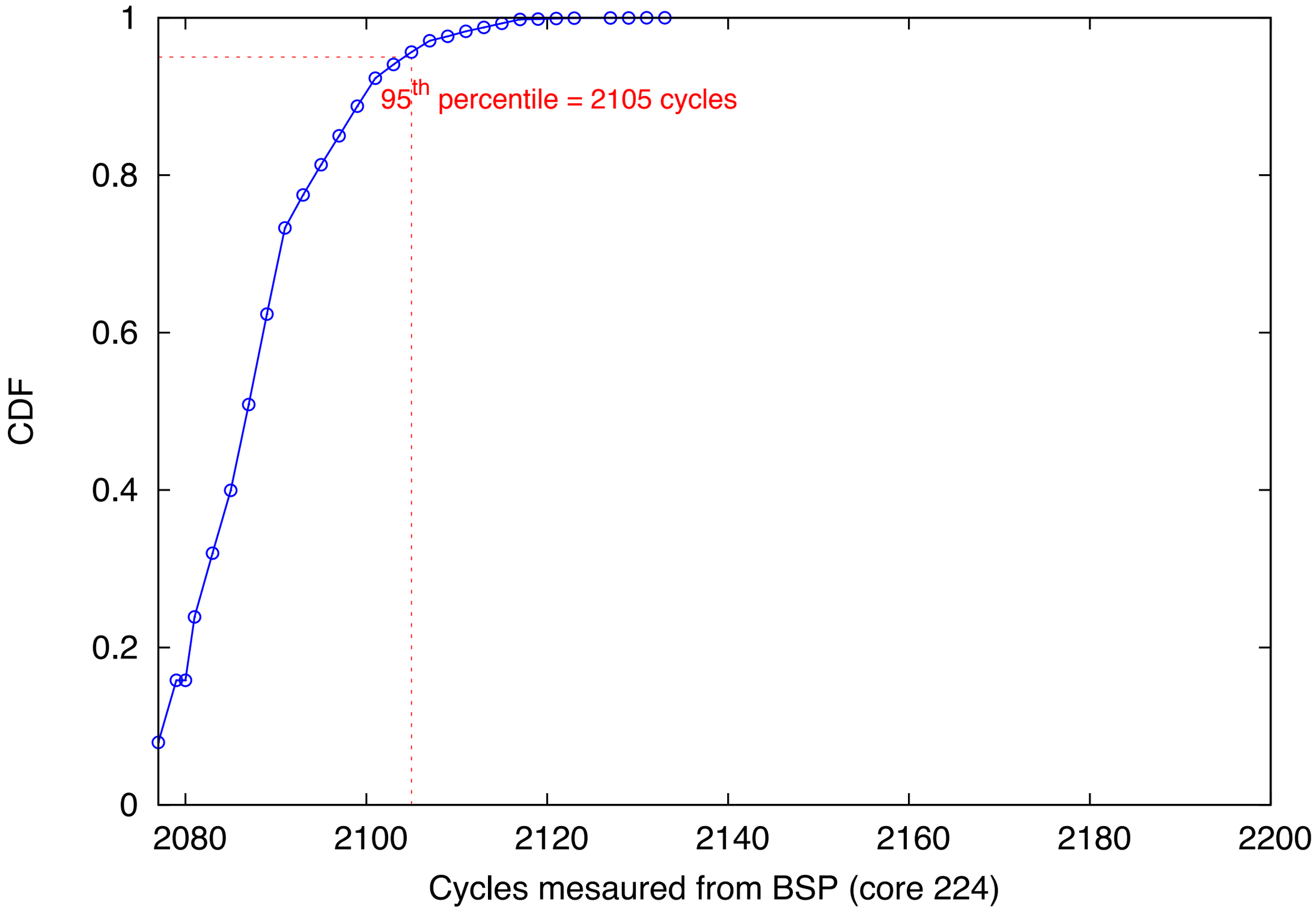
Broadcast wakeups on x64



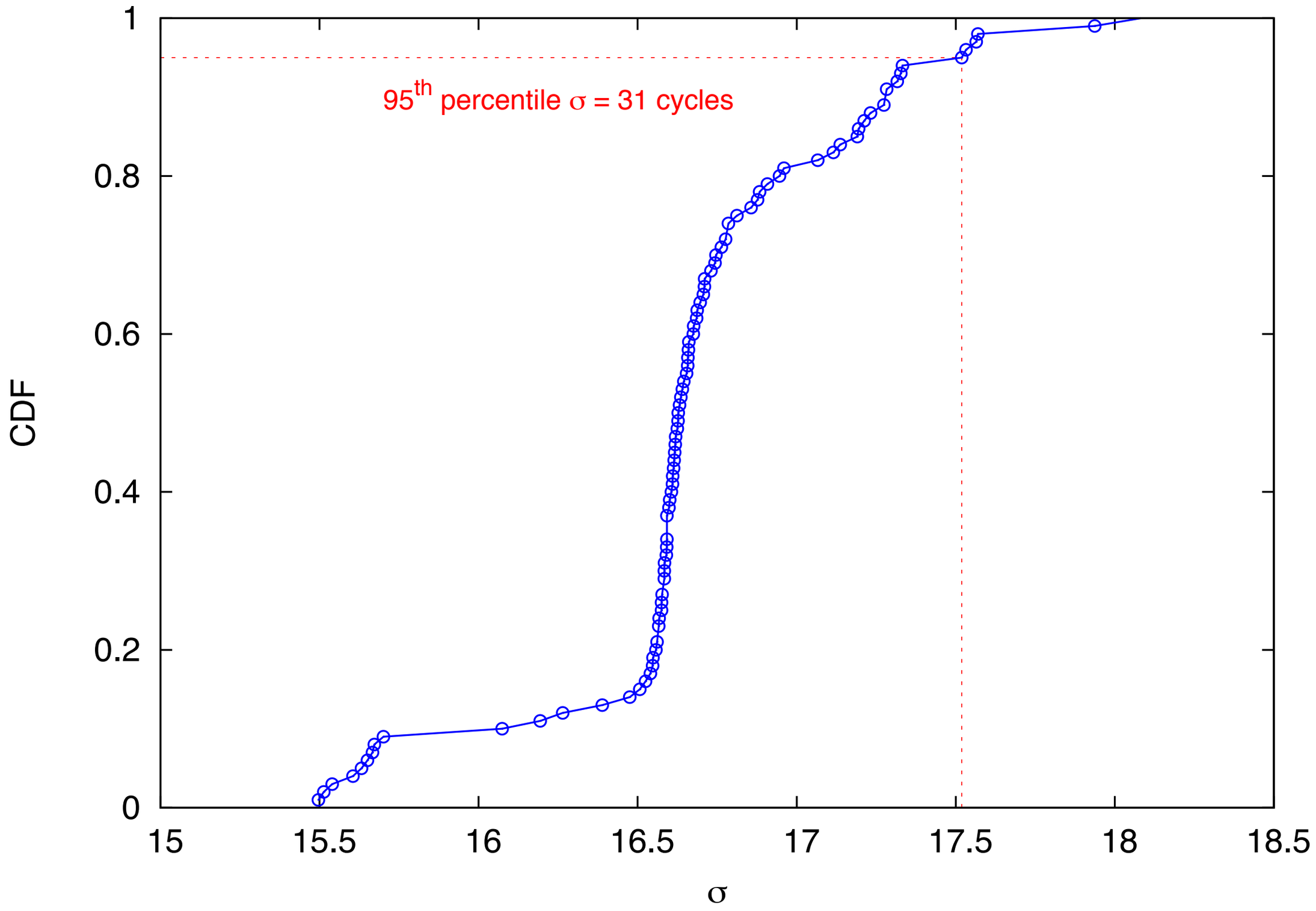
Unicast IPIs on phi



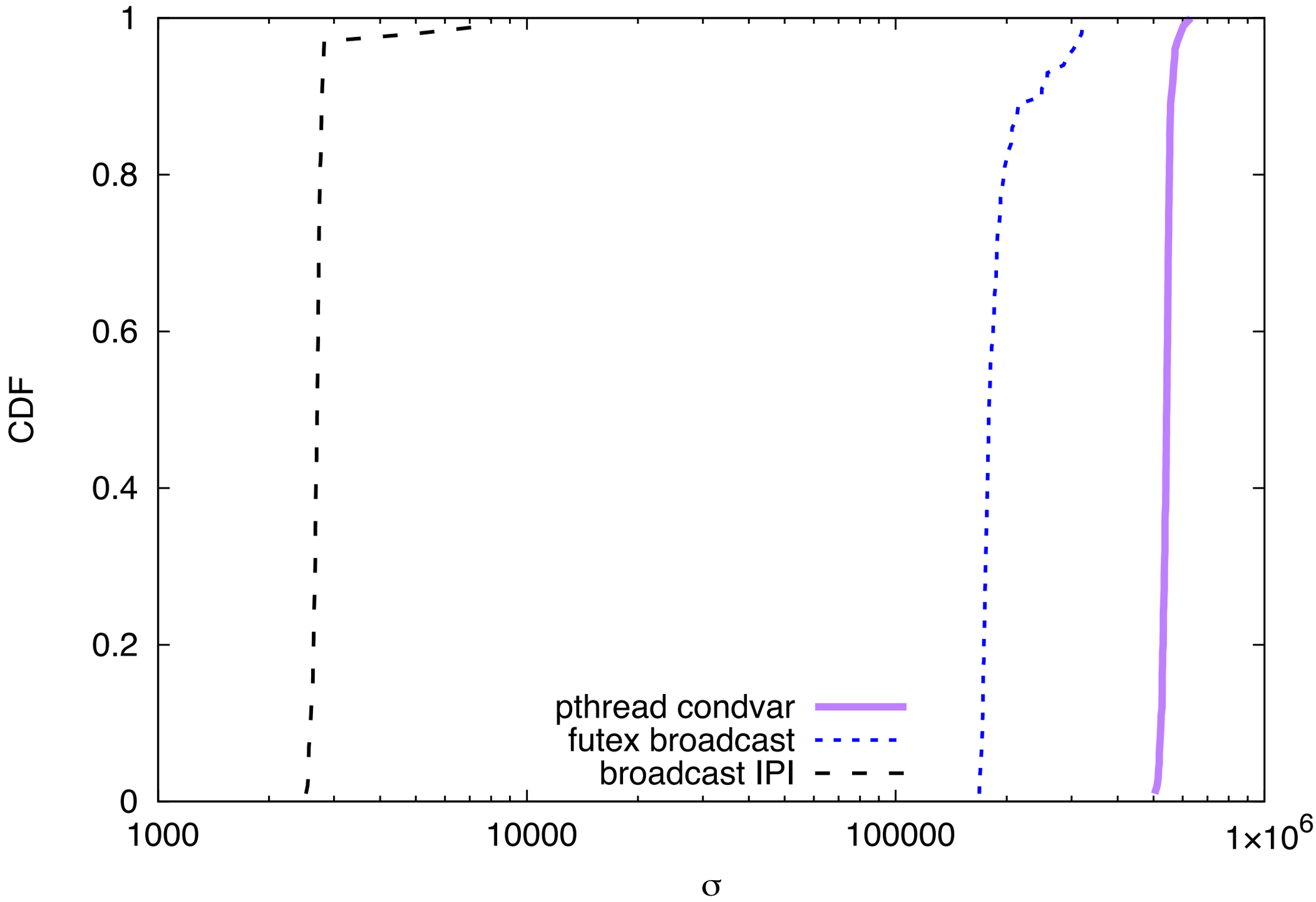
Roundtrip IPIs on phi



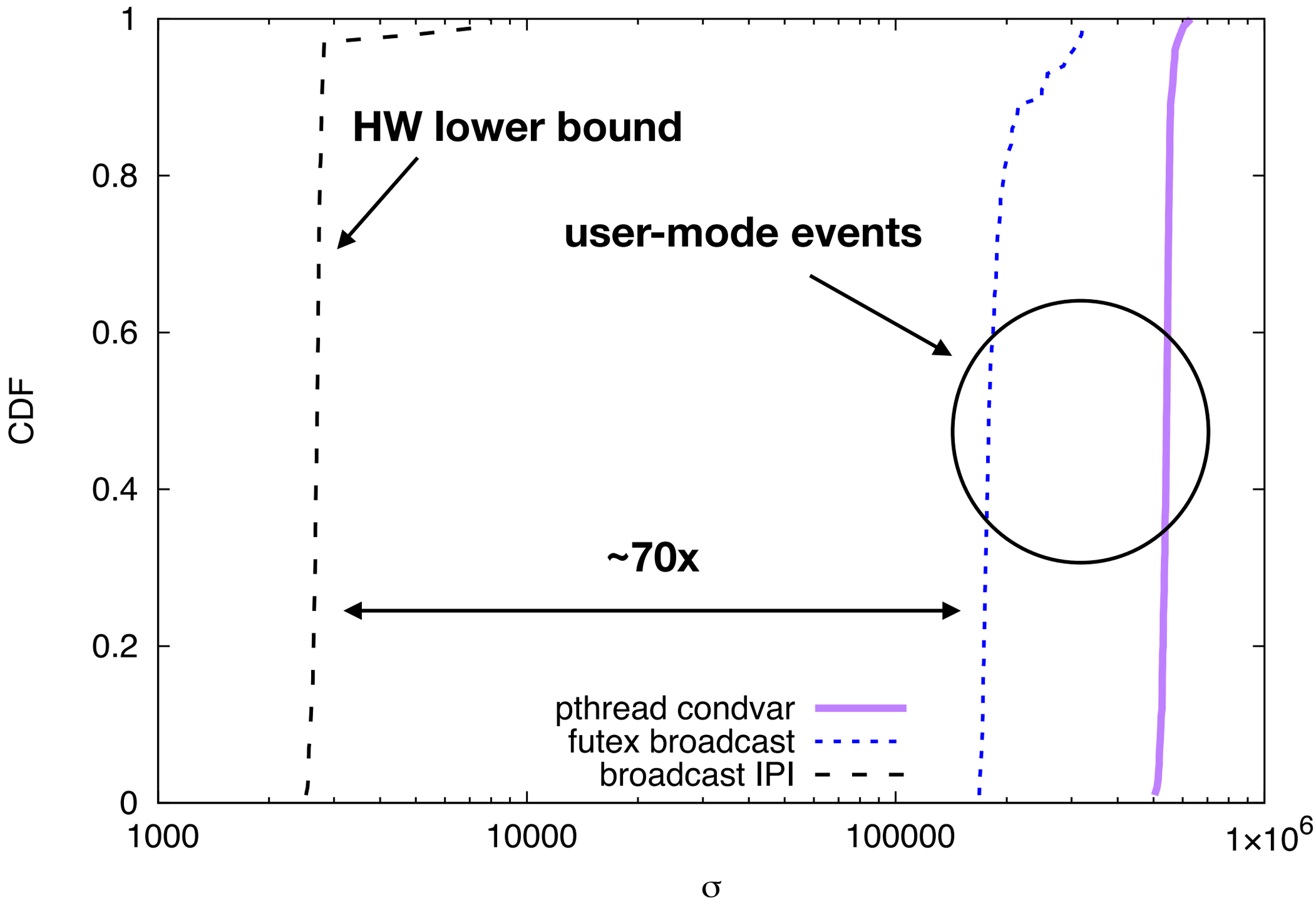
Multicast IPIs on phi



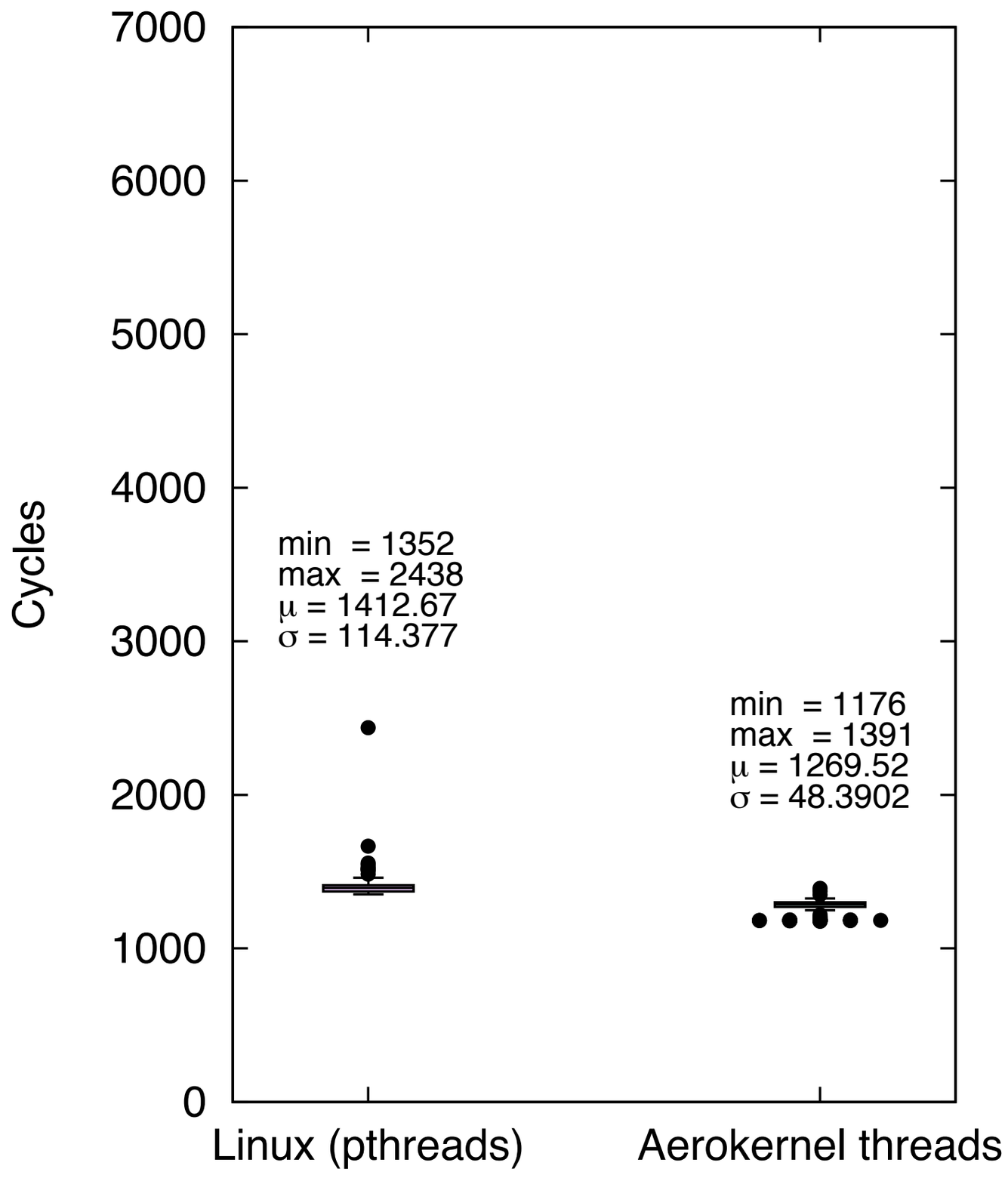
Wakeup deviation on x64



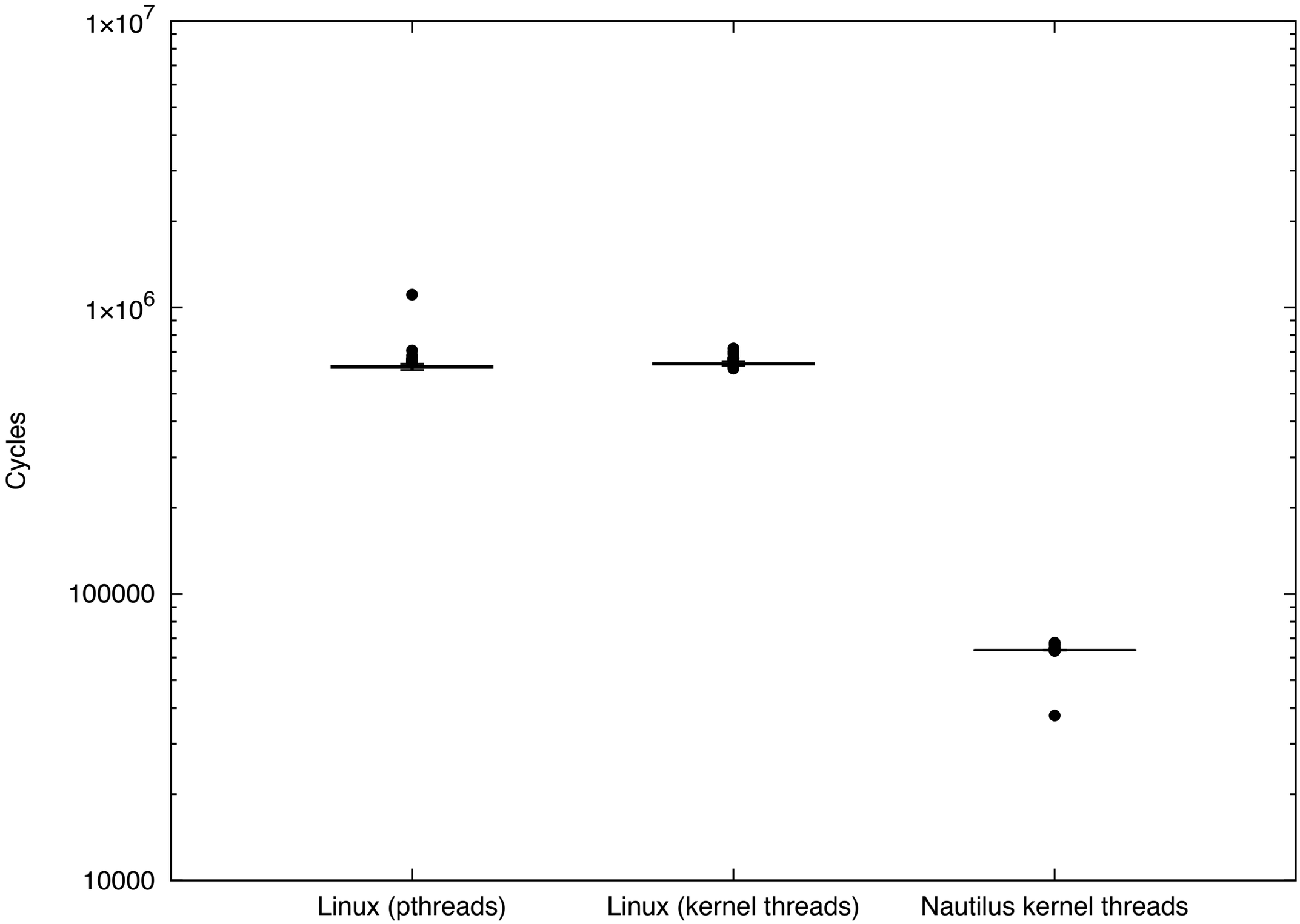
Wakeup deviation on x64



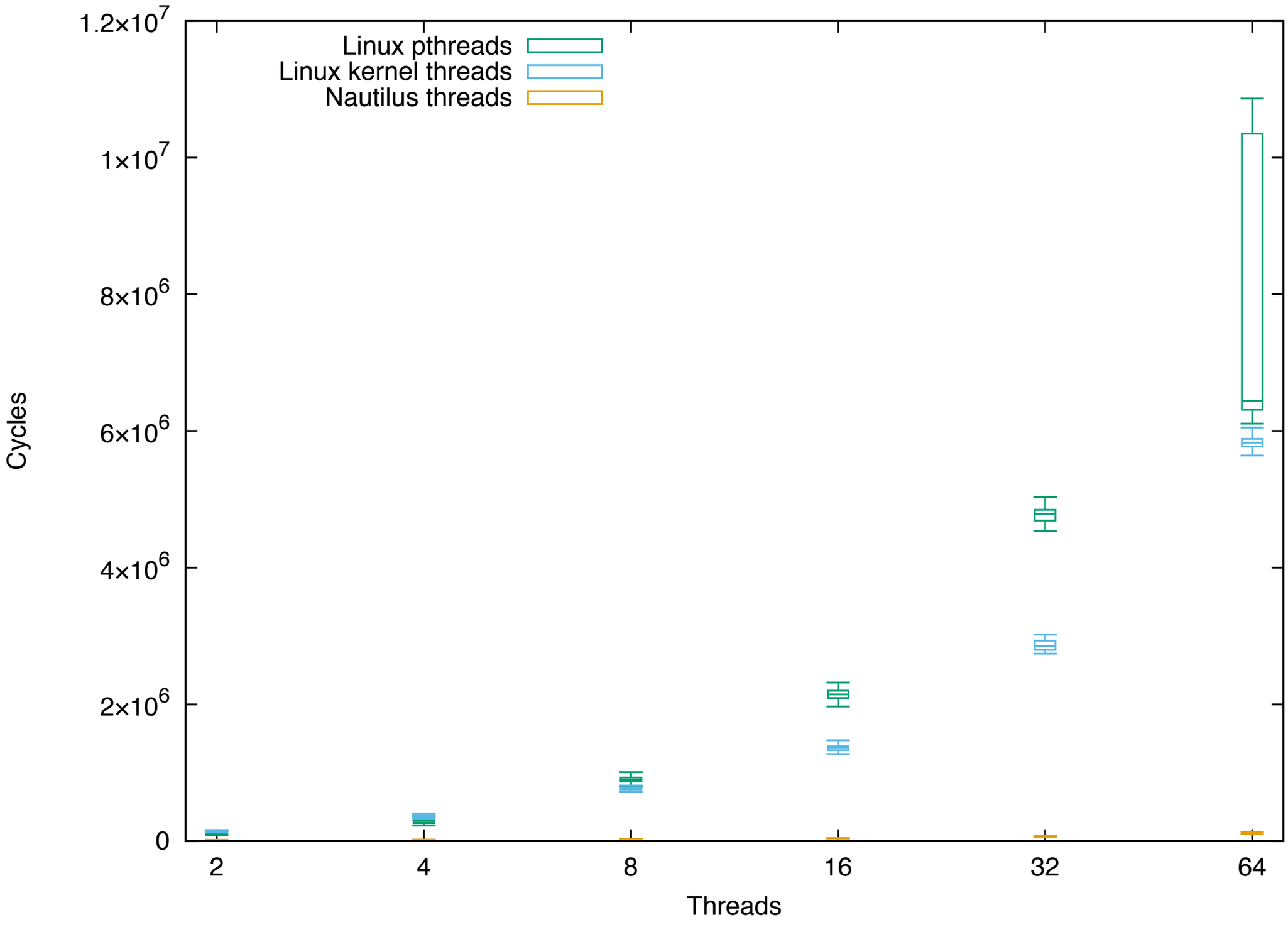
thread context switch on x64



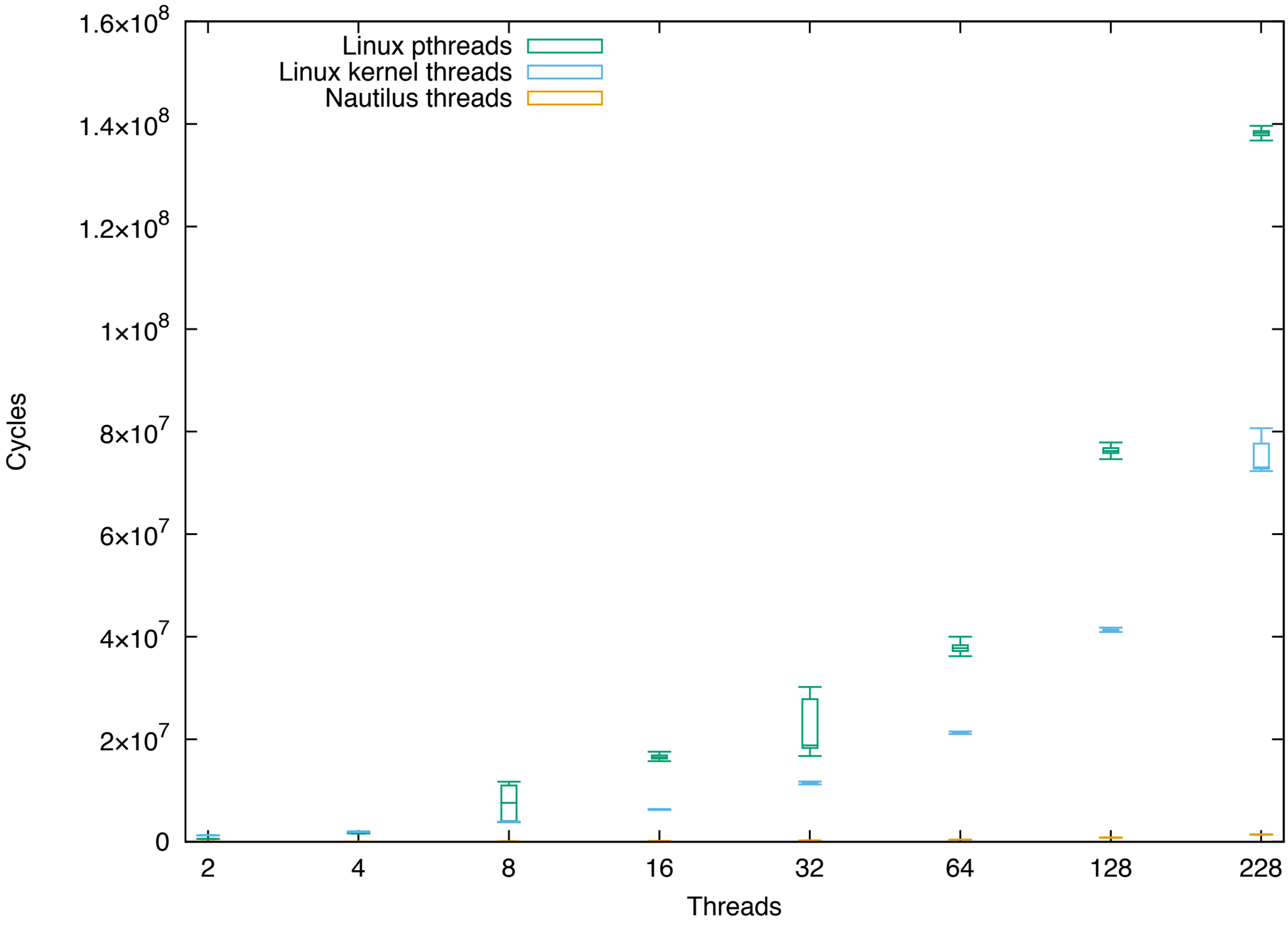
thread create + launch on phi



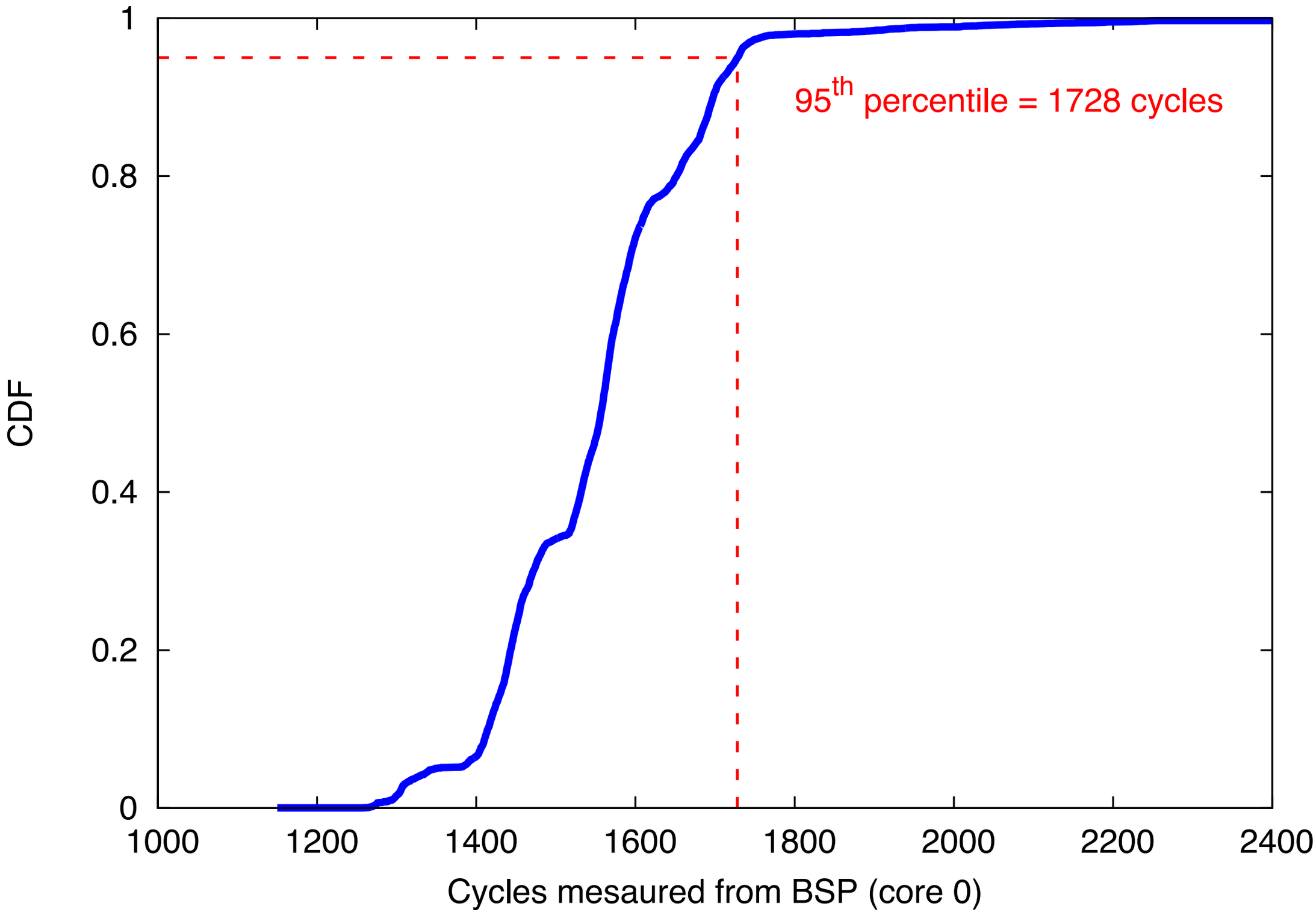
thread create + launch (many threads) on x64



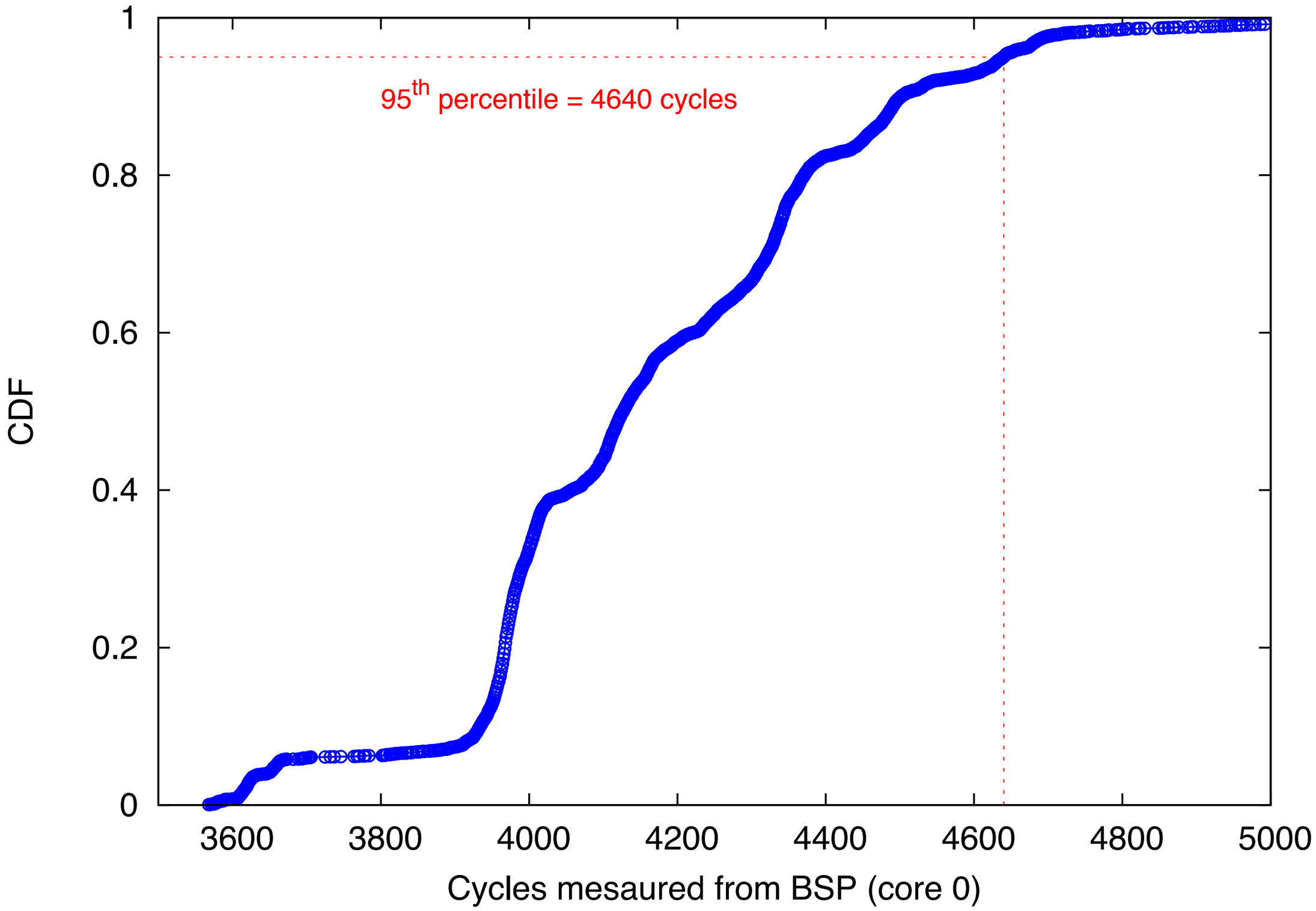
thread create + launch (many threads) on phi



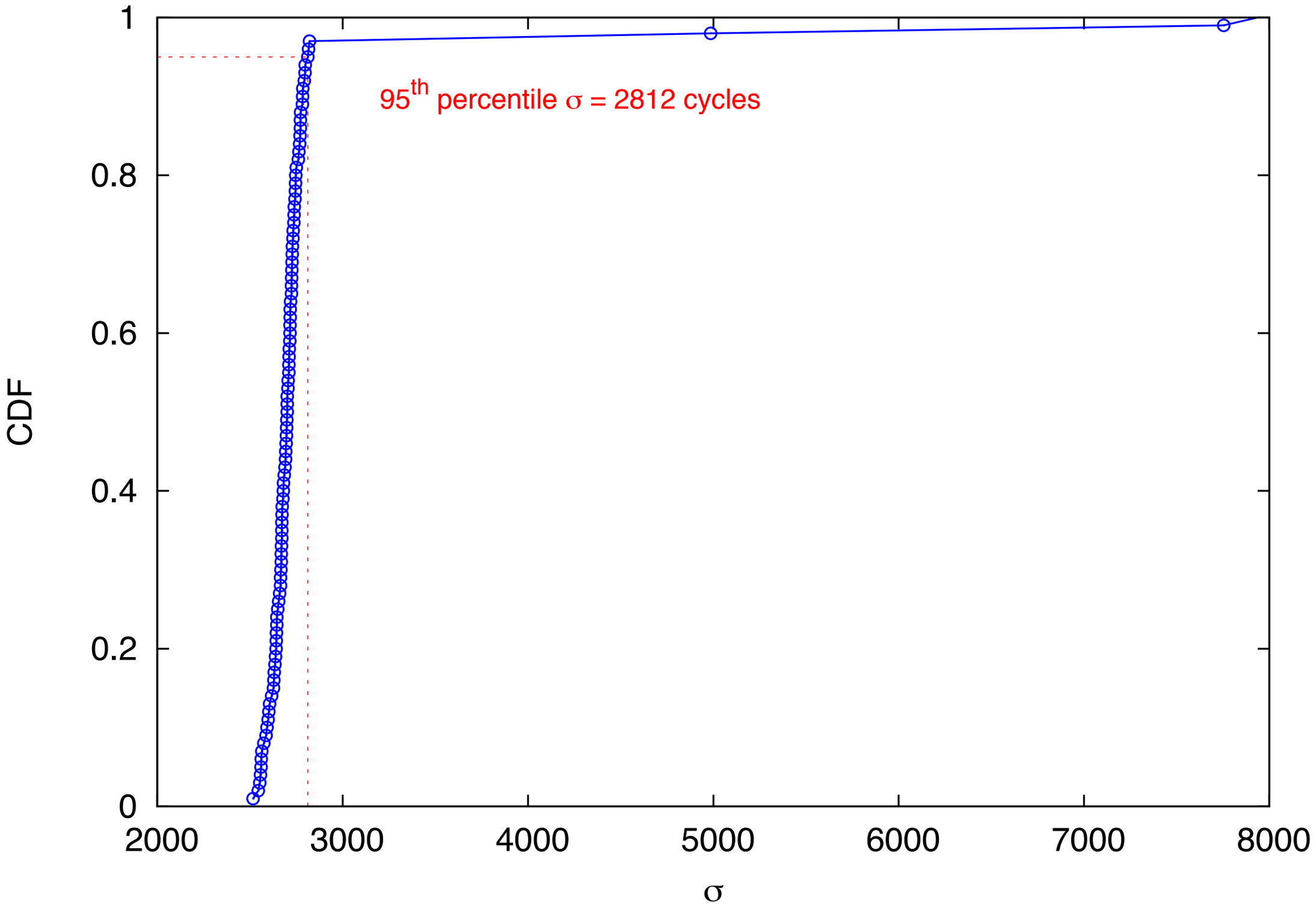
Unicast IPIs on x64



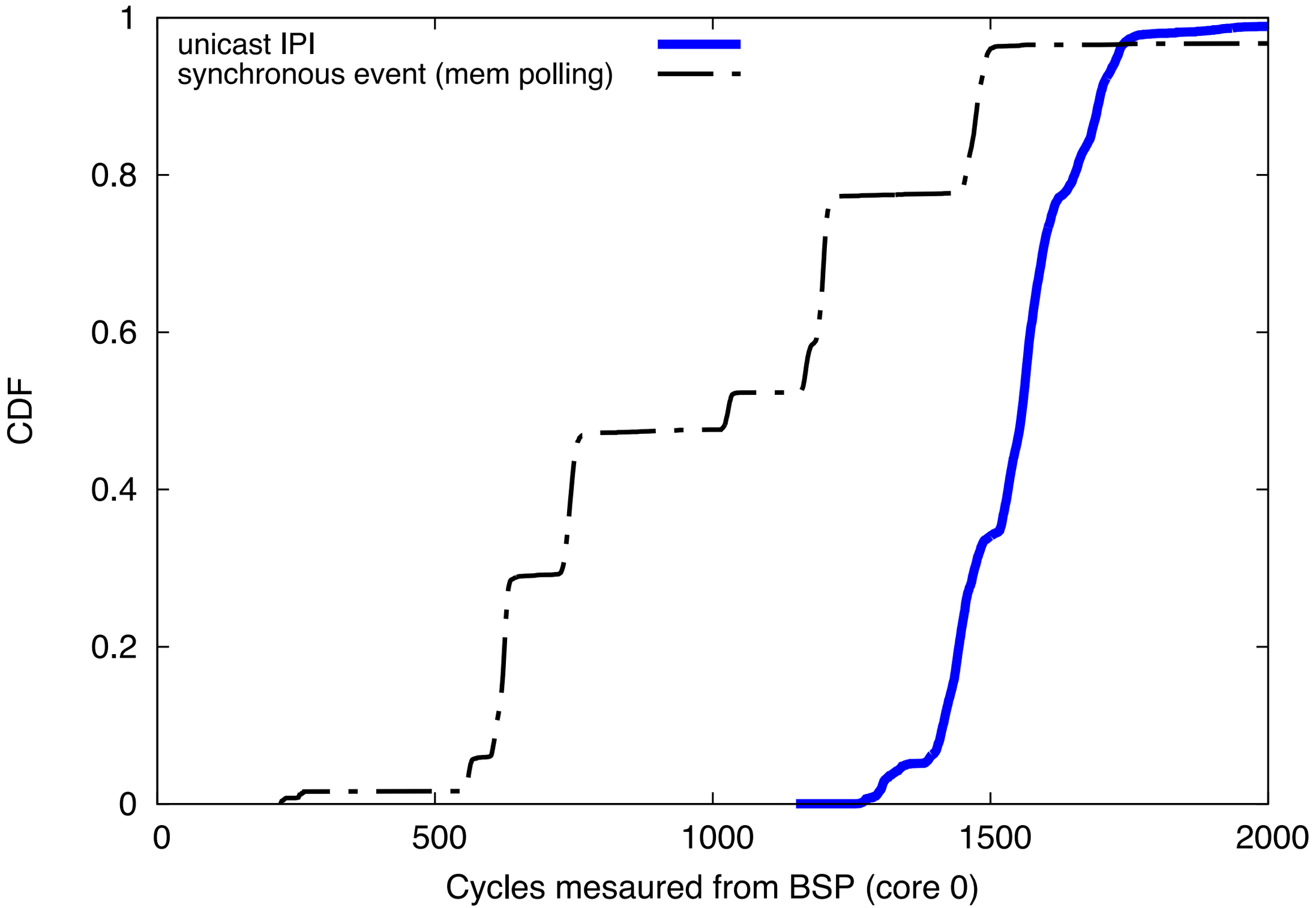
Roundtrip IPIs on x64



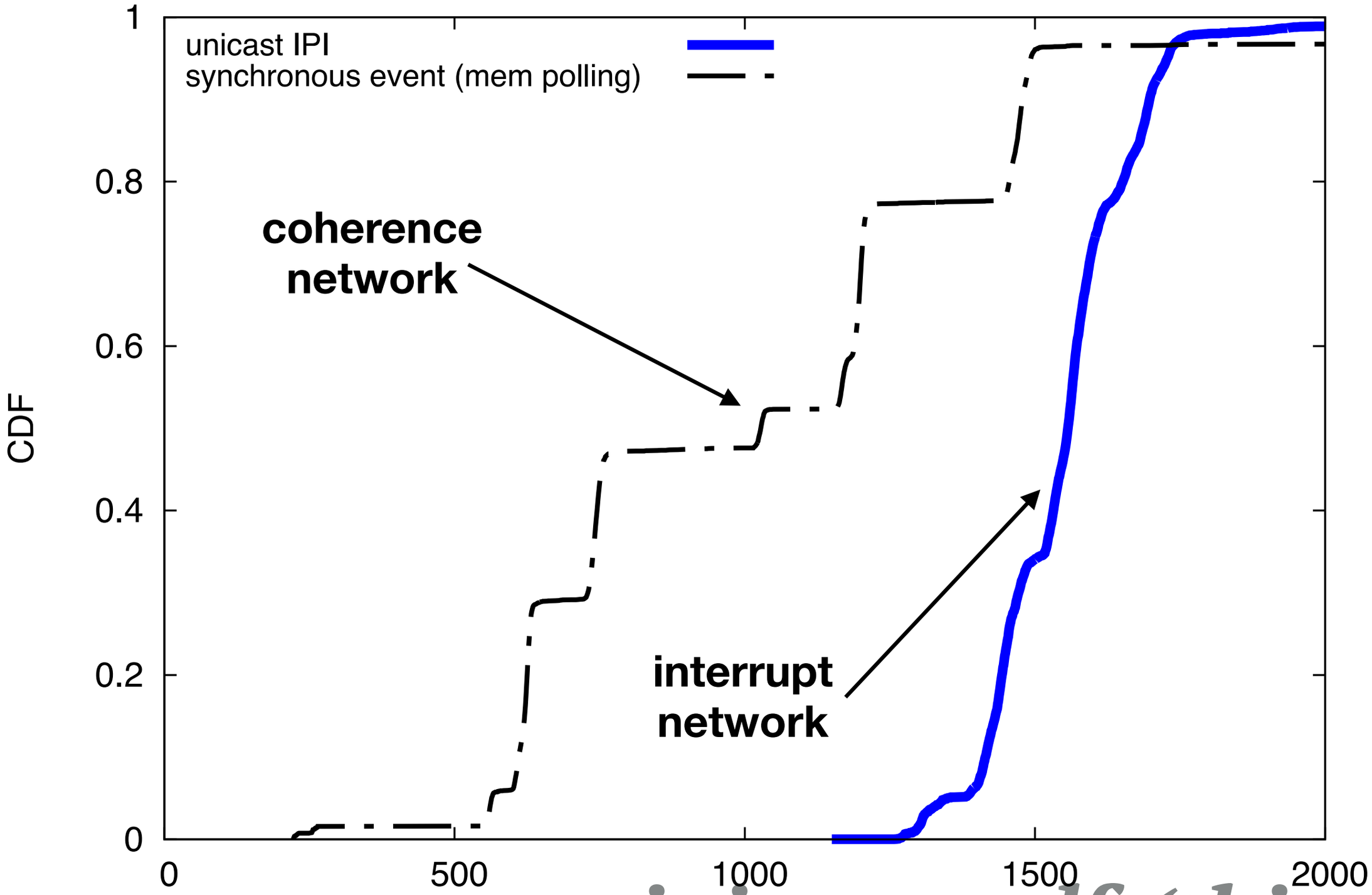
Multicast IPIs on x64



Unicast IPI vs memory polling

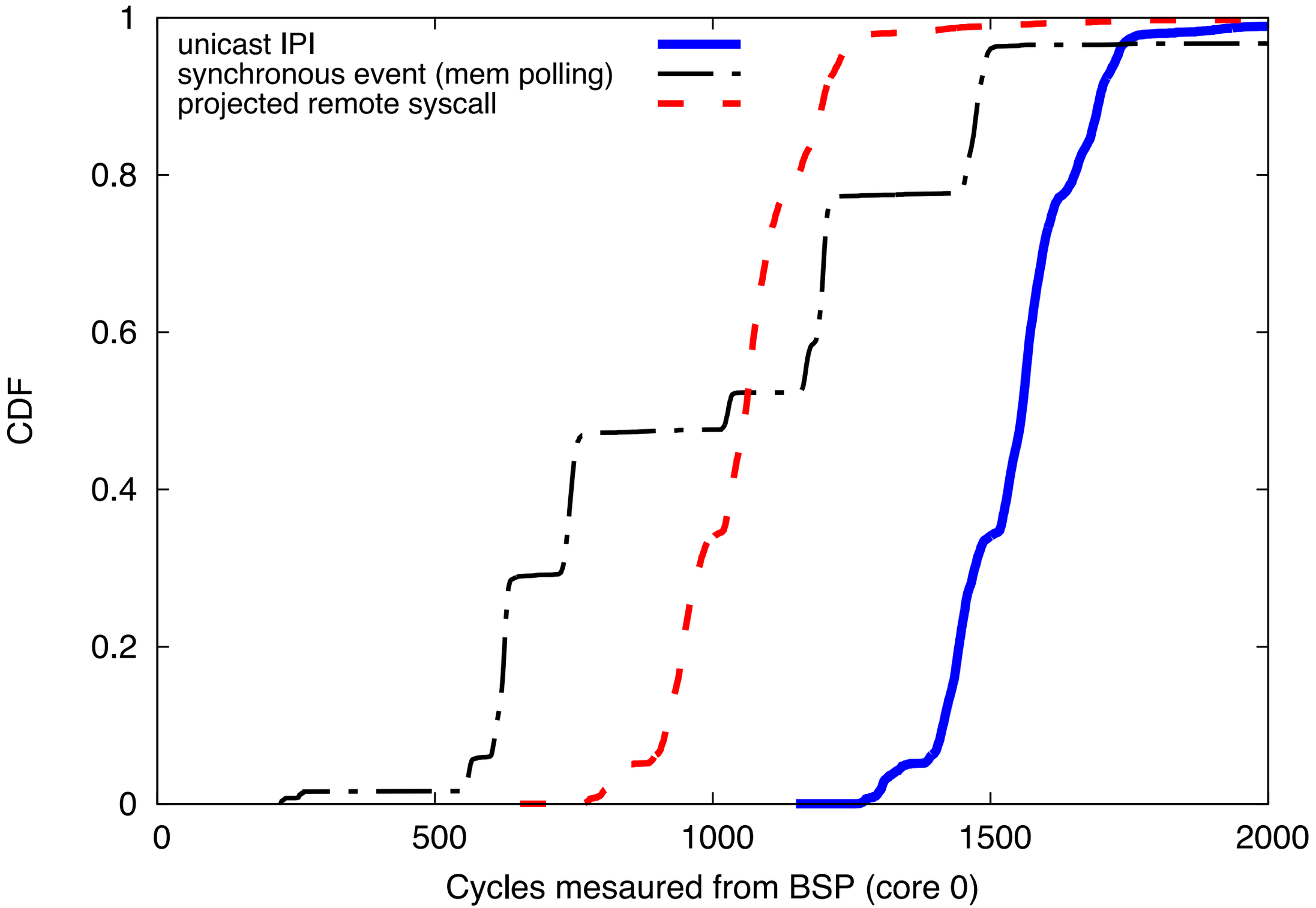


Unicast IPI vs memory polling

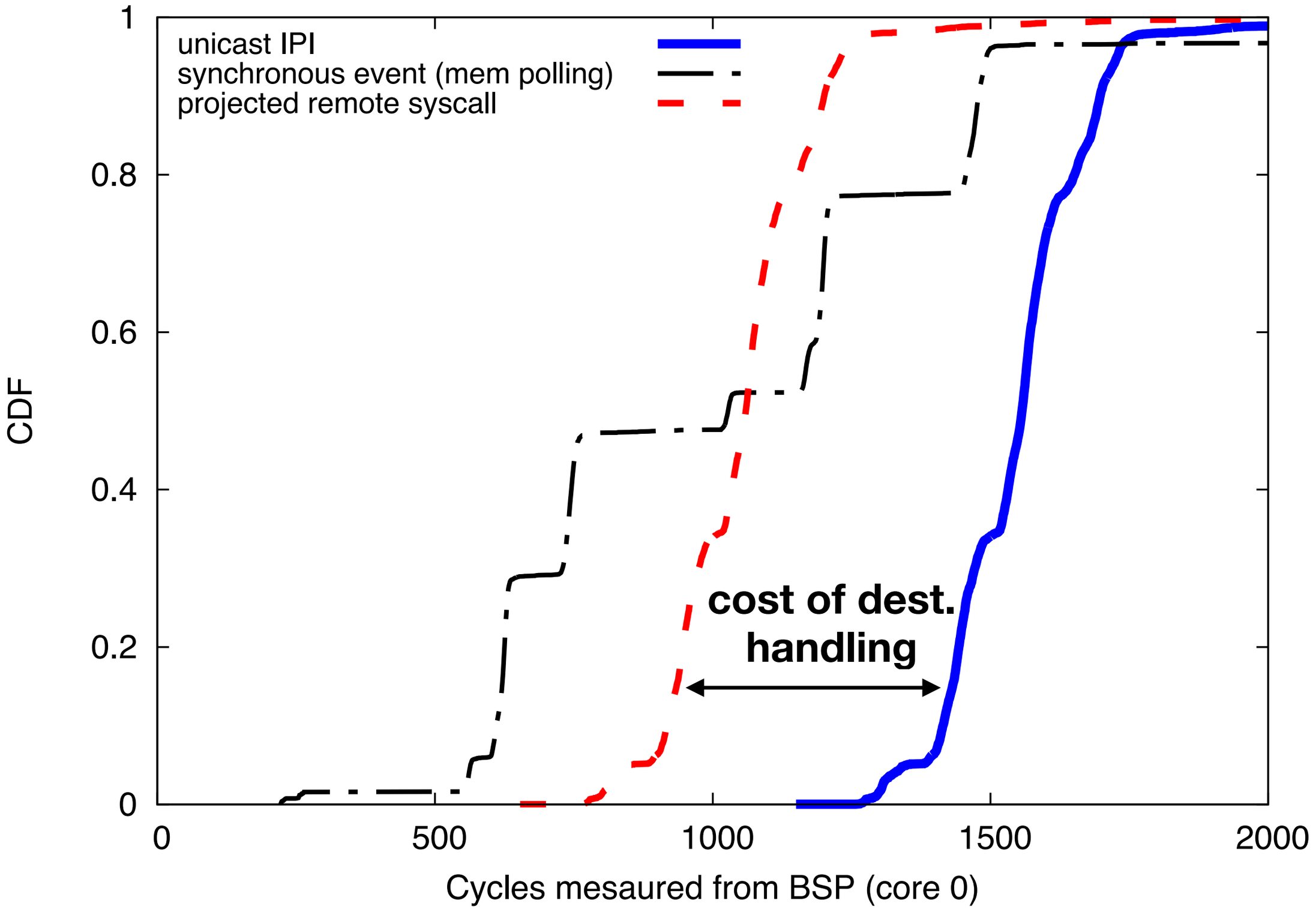


ipi sync cdf (this one annotations)

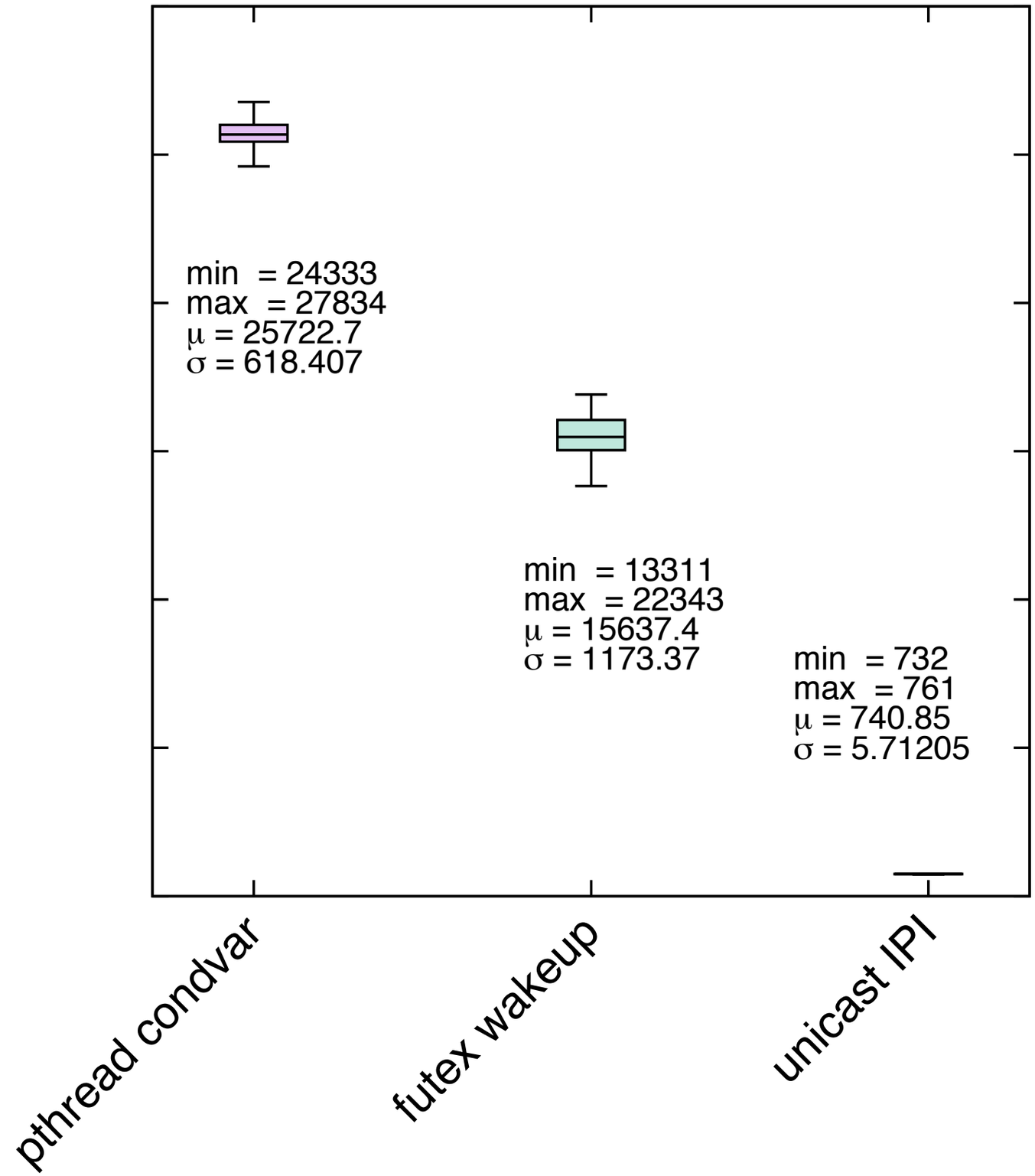
Unicast IPI vs memory polling



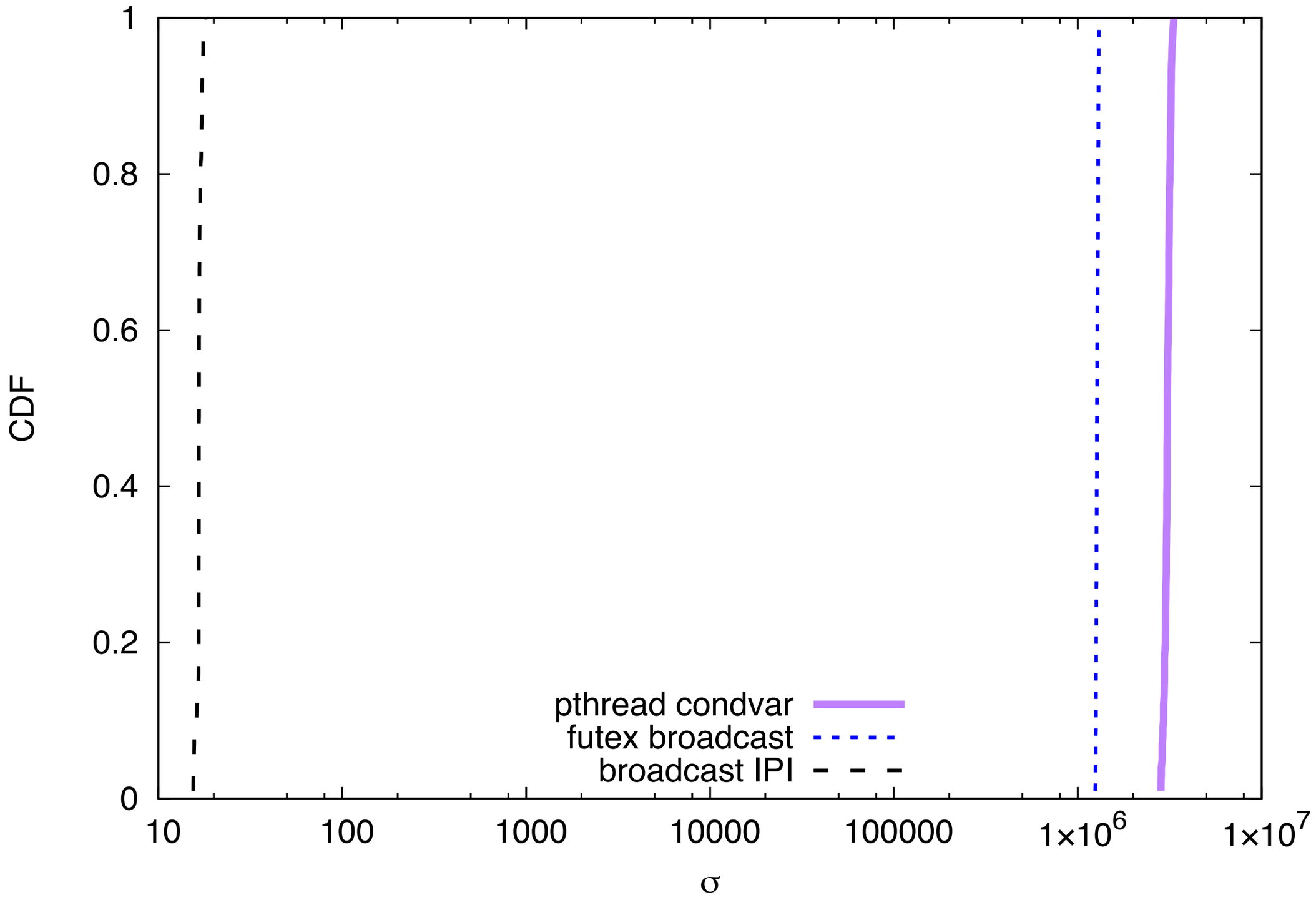
Unicast IPI vs memory polling



Single wakeup on phi

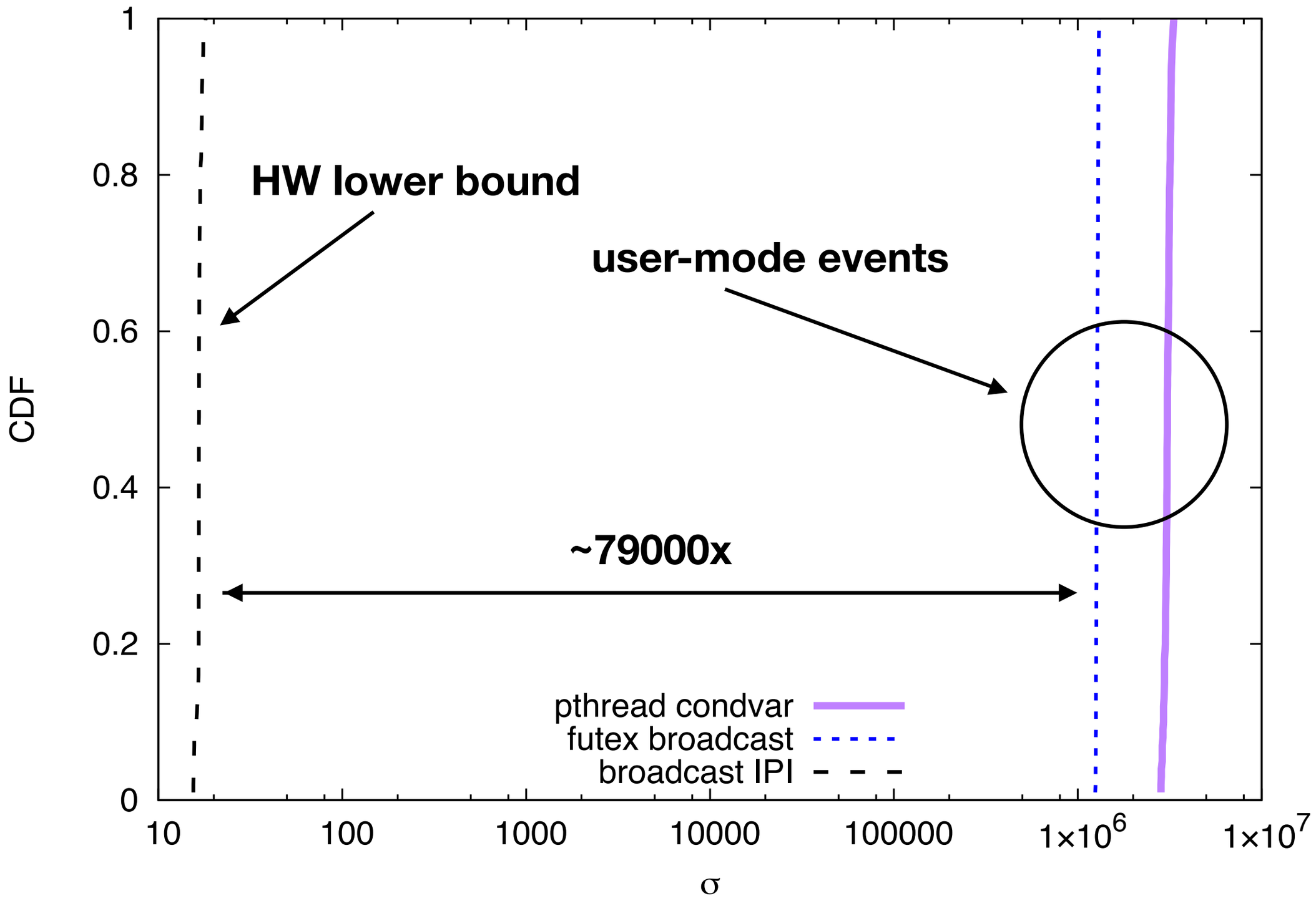


Wakeup deviation on phi

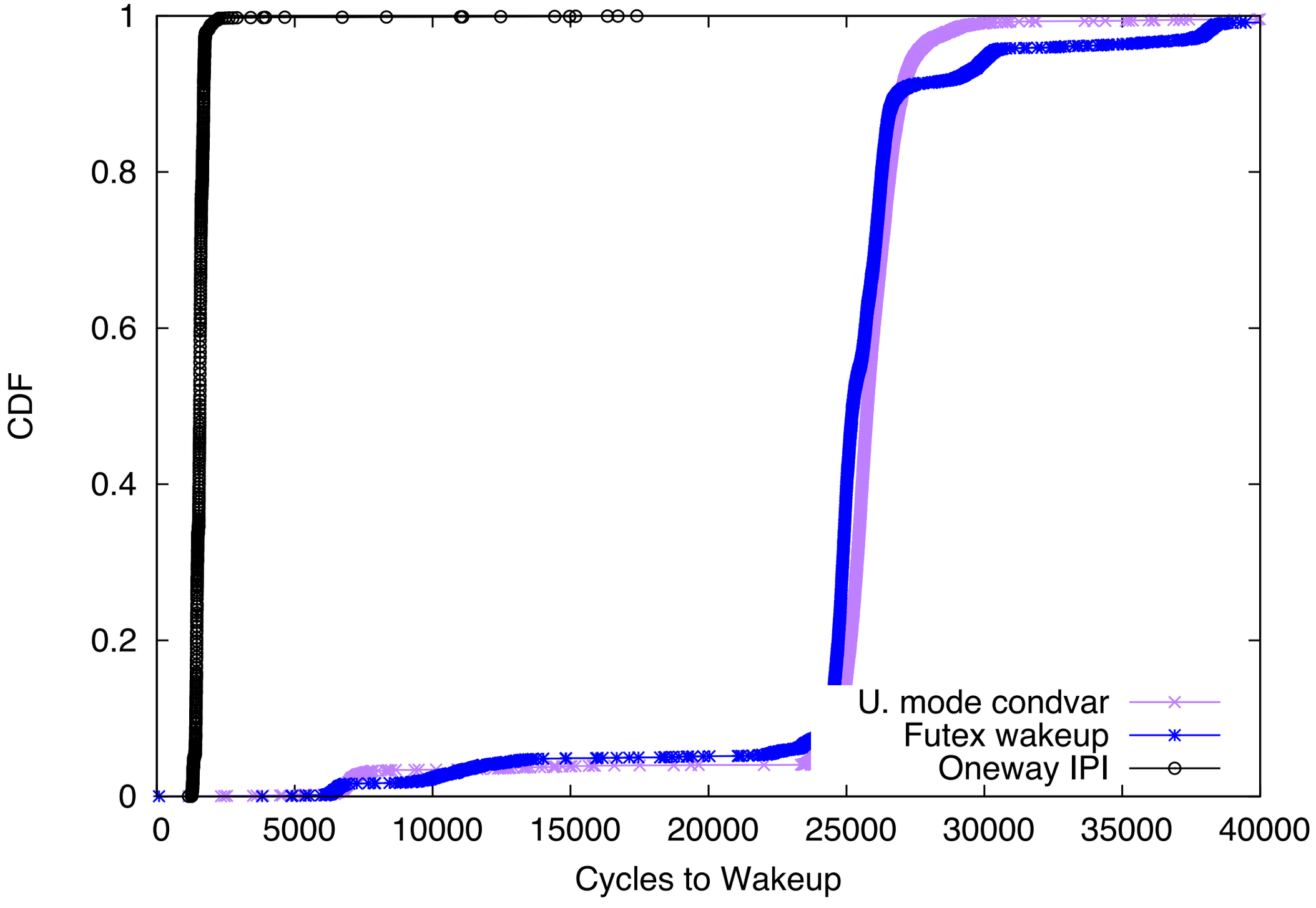


wakeup phi many

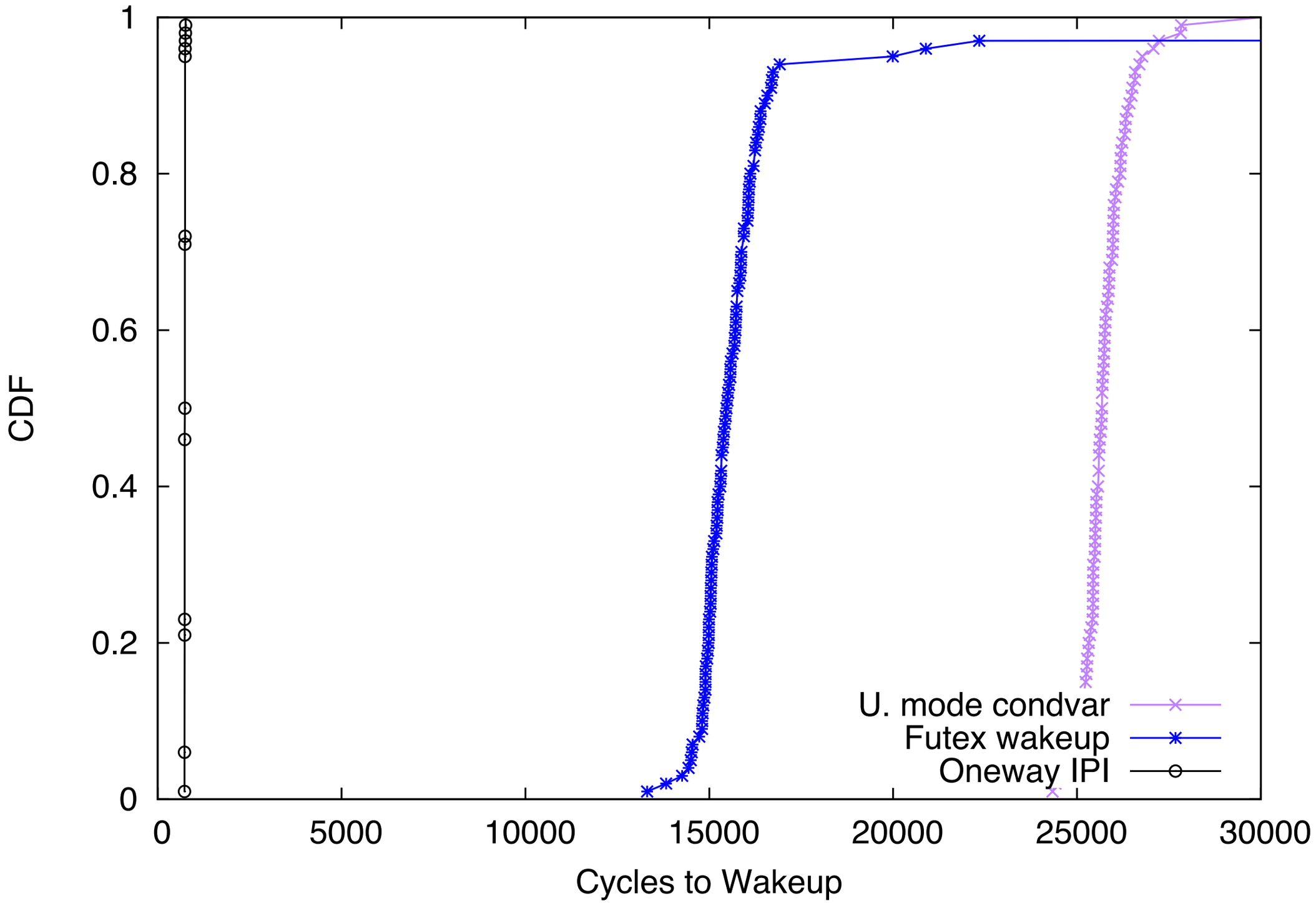
Wakeup deviation on phi



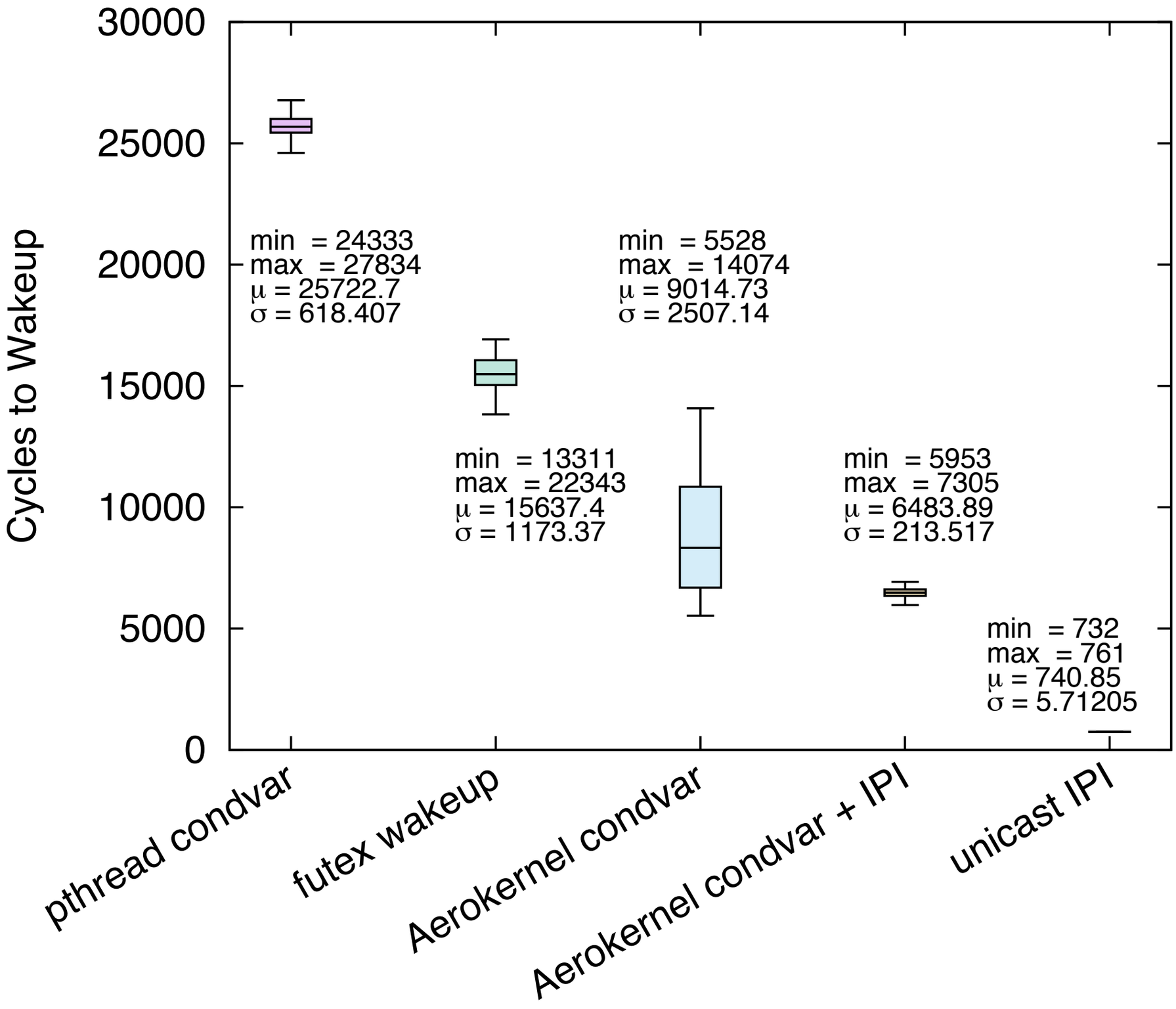
Single wakeup on x64 (CDF)



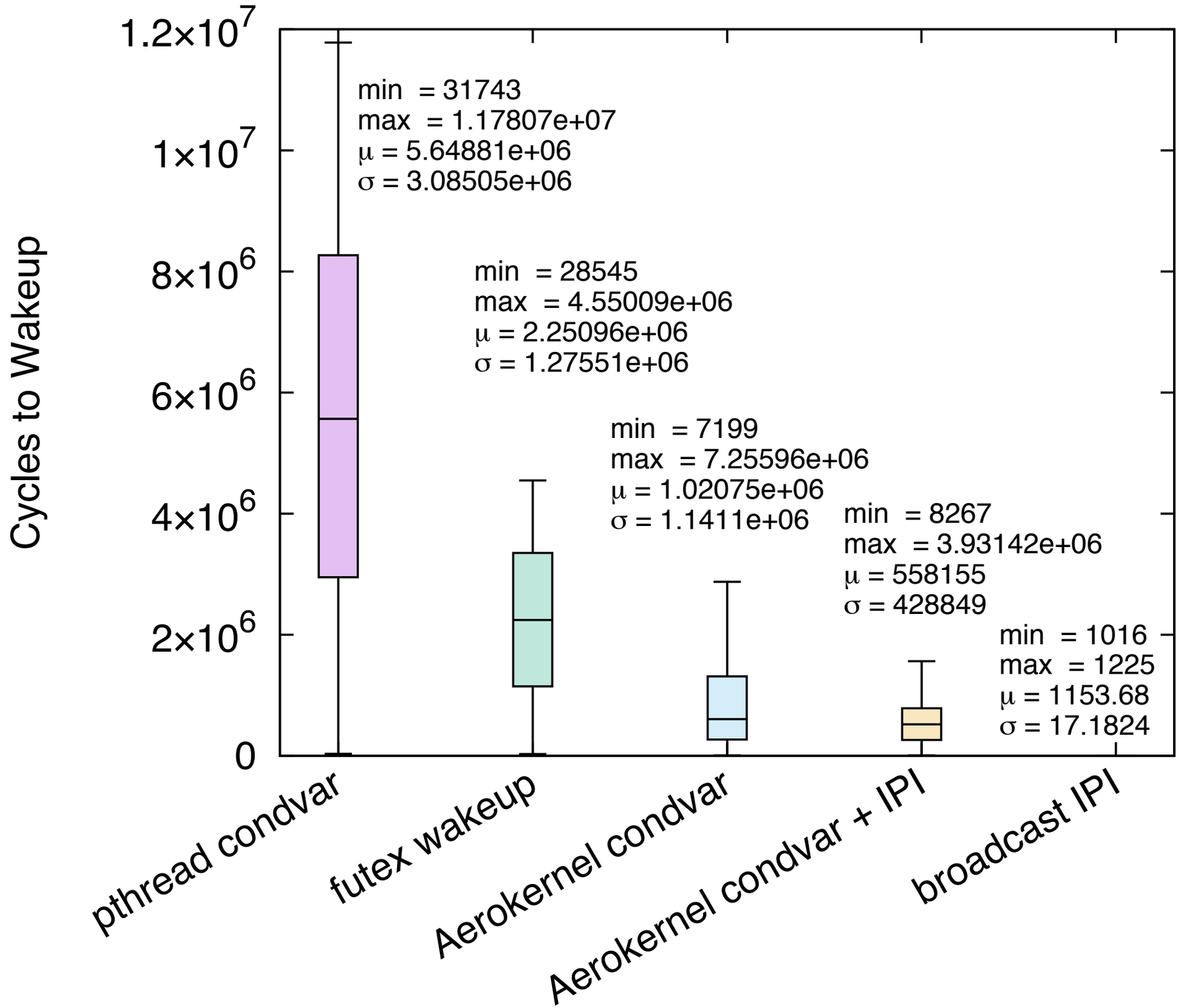
Single wakeup on phi (CDF)



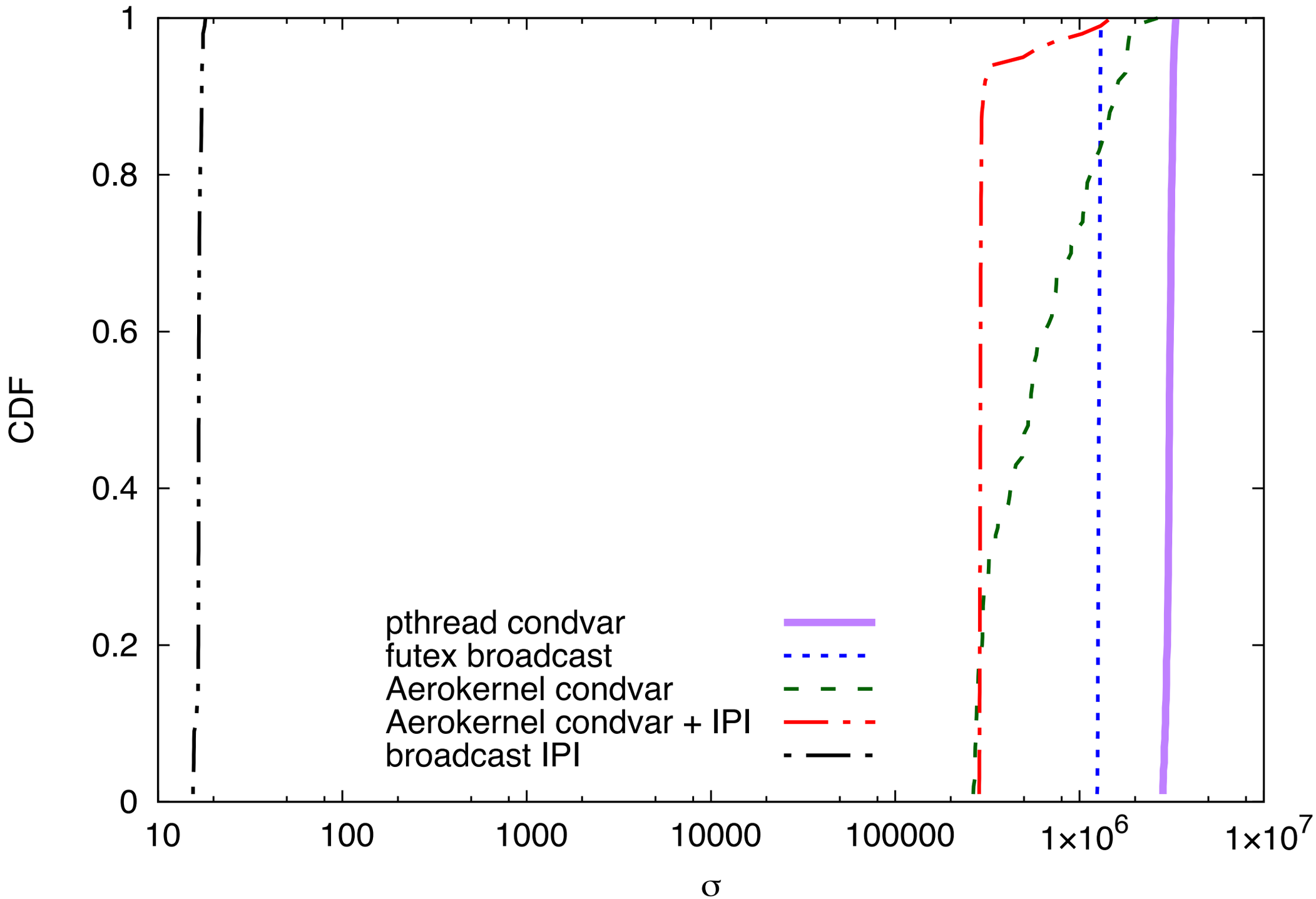
Single wakeup on phi



Many wakeups on phi

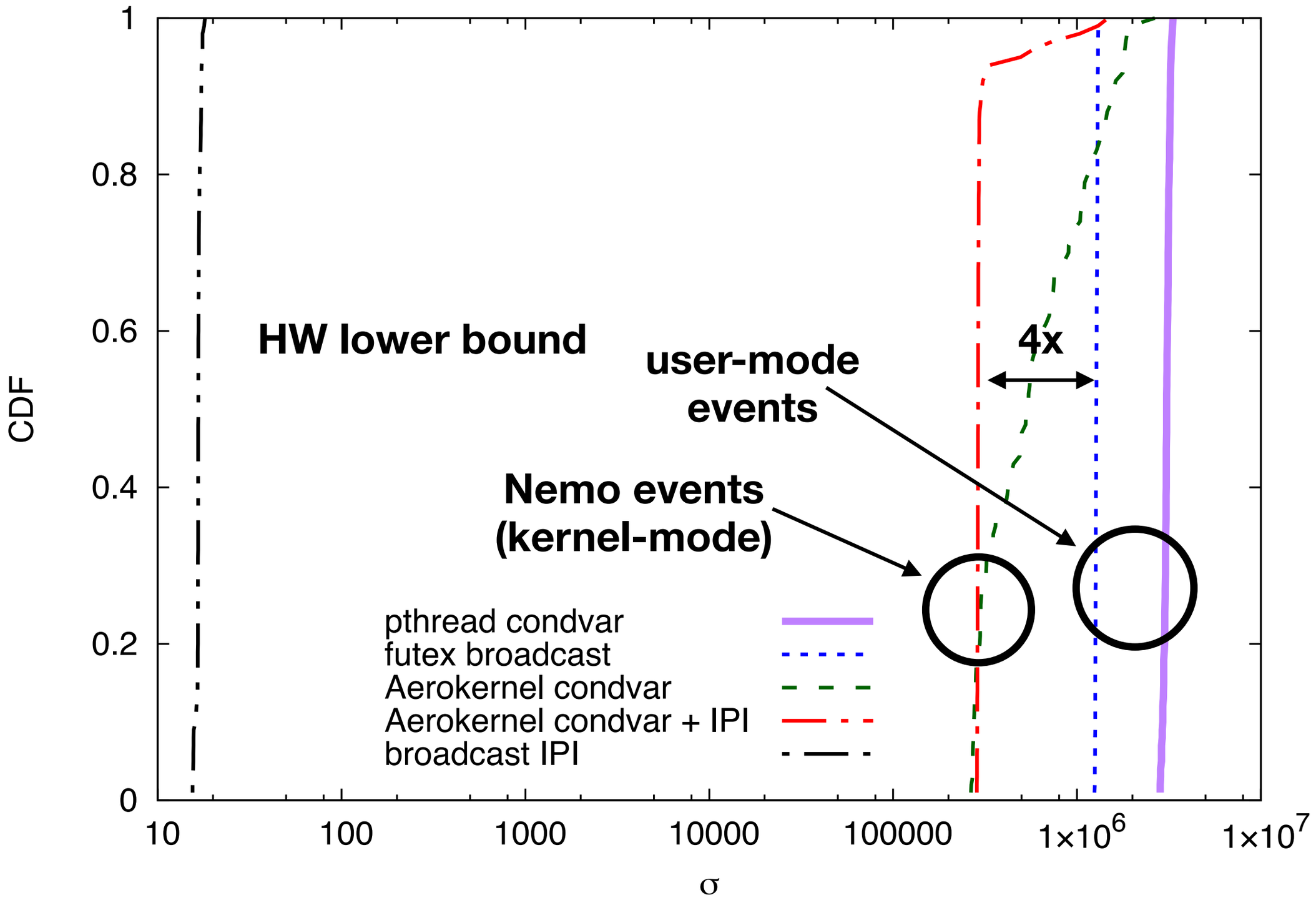


Wakeup deviation on phi

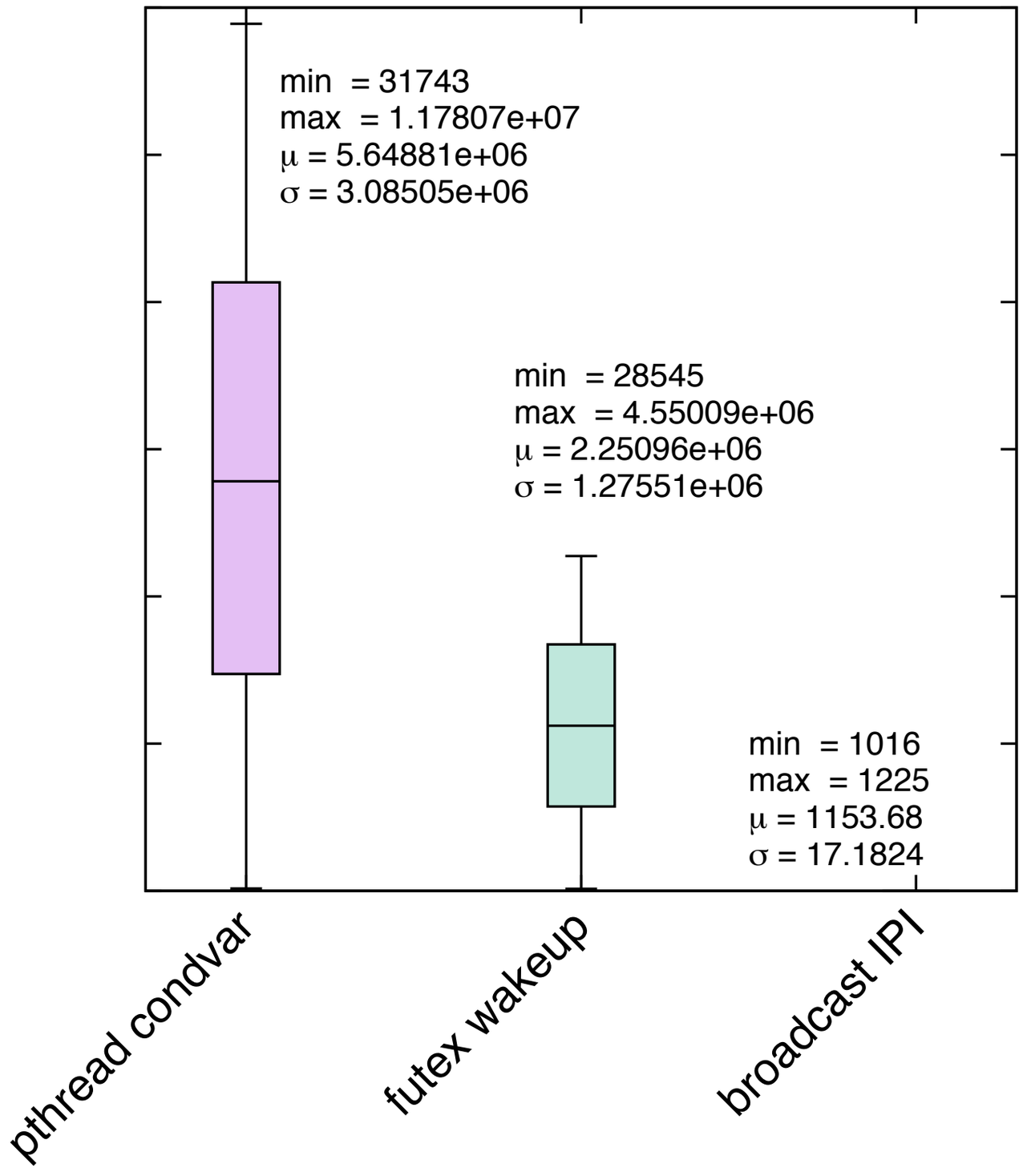


wakeup phi many wk

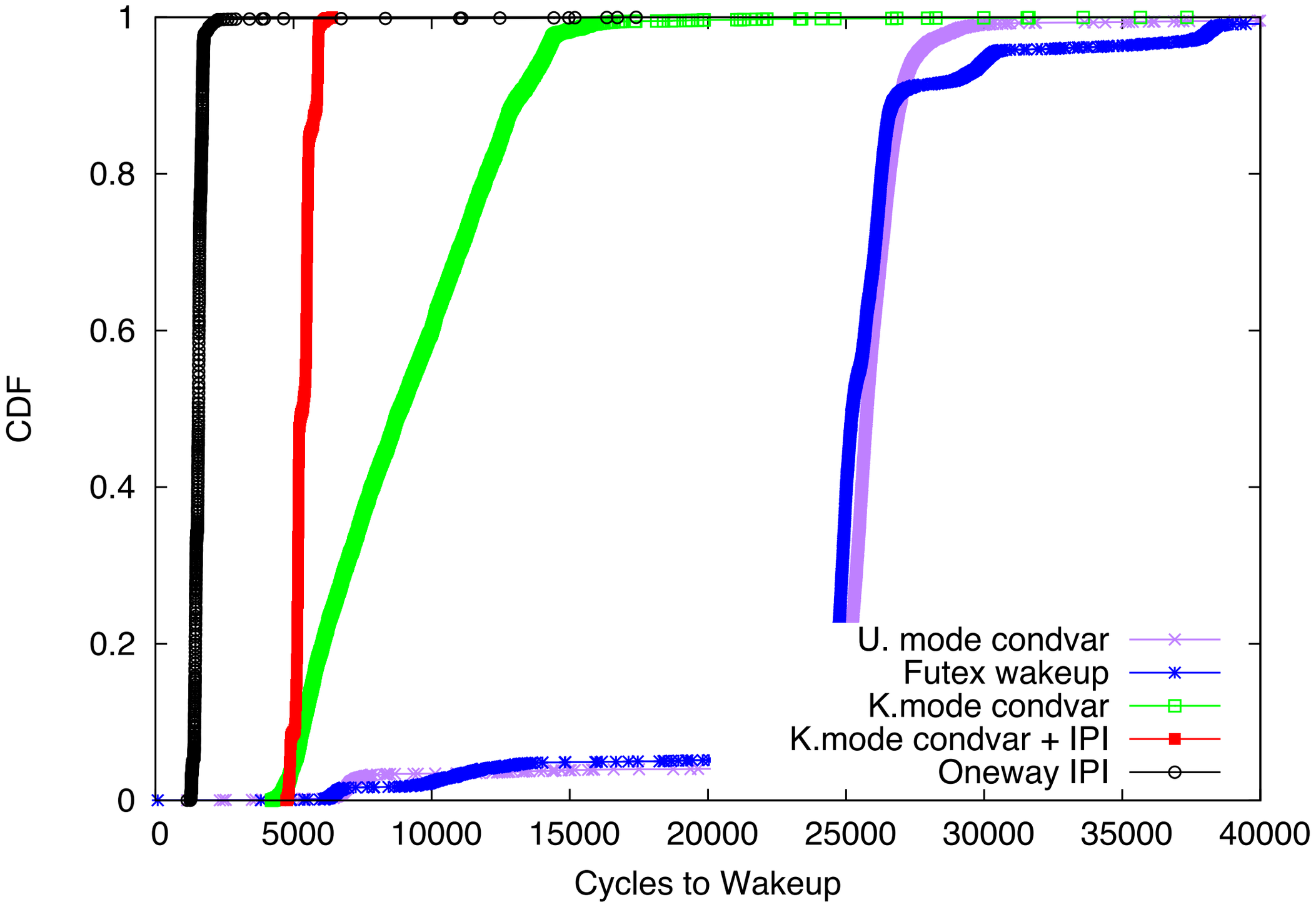
Wakeup deviation on phi



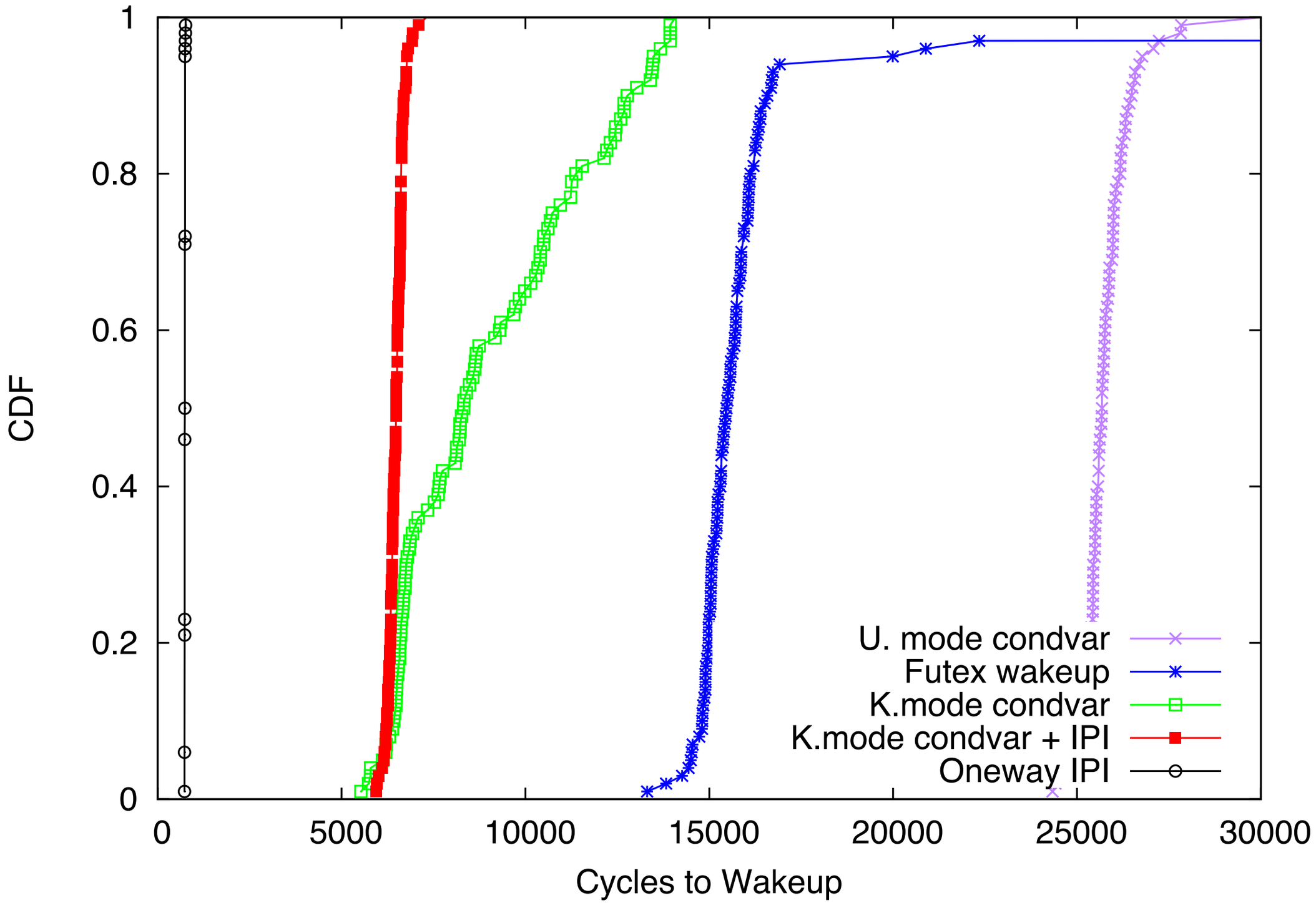
Broadcast wakeups on phi



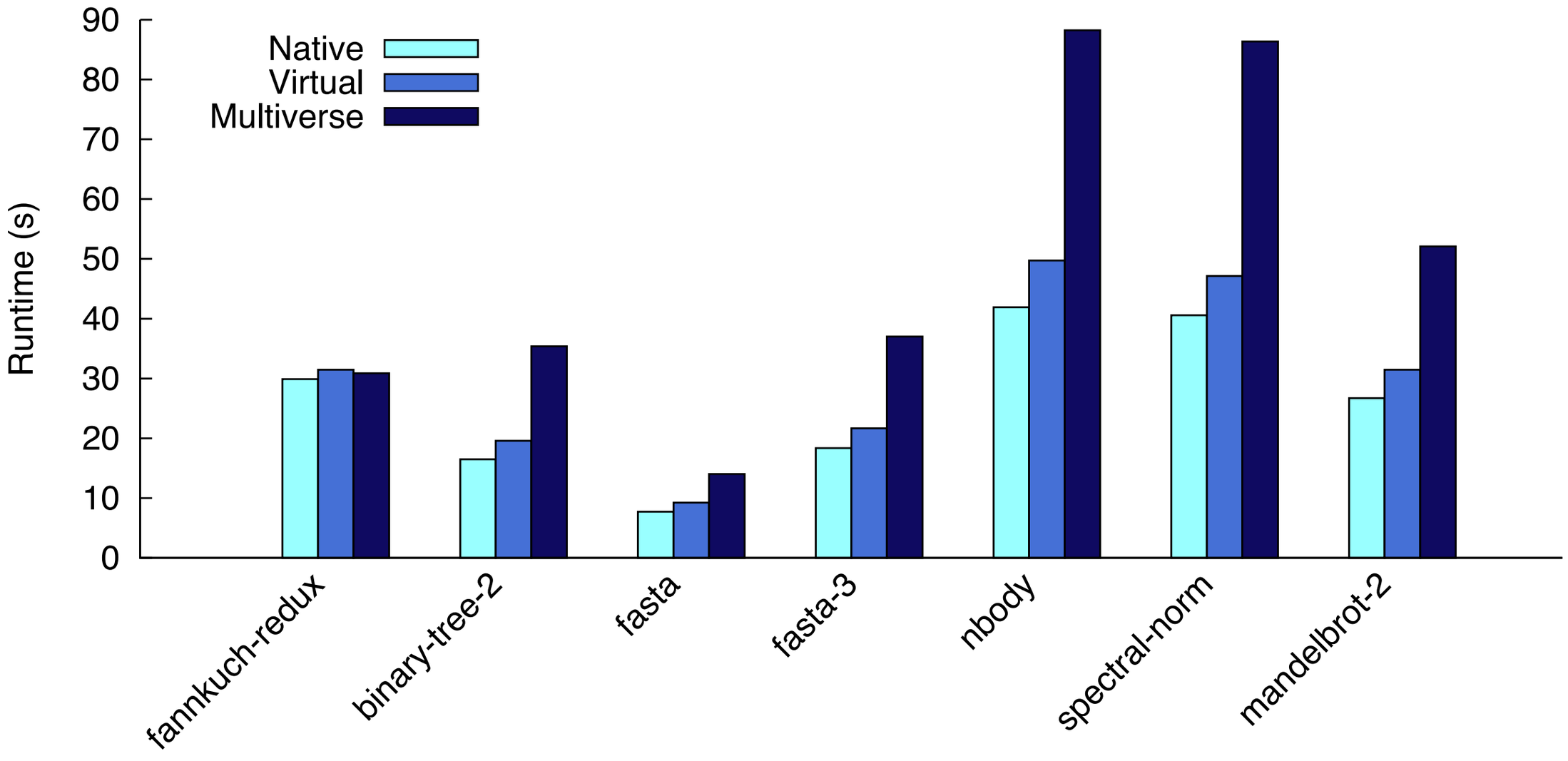
Single wakeup on x64 (CDF)



Single wakeup on phi (CDF)



racket multiverse overheads



system call overheads