ACCELERATING ASYNCHRONOUS EVENTS FOR HYBRID PARALLEL RUNTIMES

Kyle C. Hale and Peter Dinda

Northwestern University

HOBBES

xstack.sandia.gov/hobbes

ILLINOIS INSTITUTE OF TECHNOLOGY

HOBBES

xstack.sandia.gov/hobbes

nautilus.halek.co

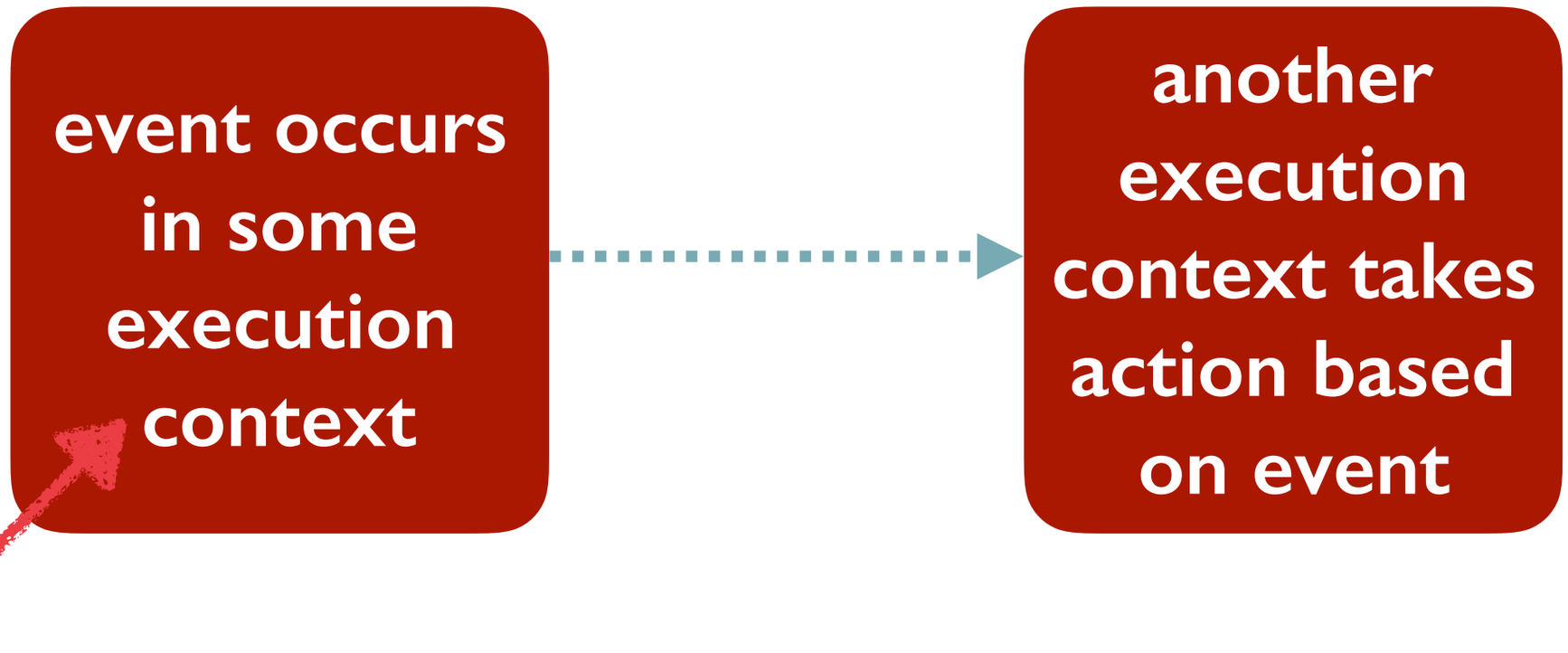
v3vee.org

Palacios

An OS Independent Embeddable VMM

v3vee.org/palacios

SOFTWARE EVENTS



SOME TYPES OF EVENTS

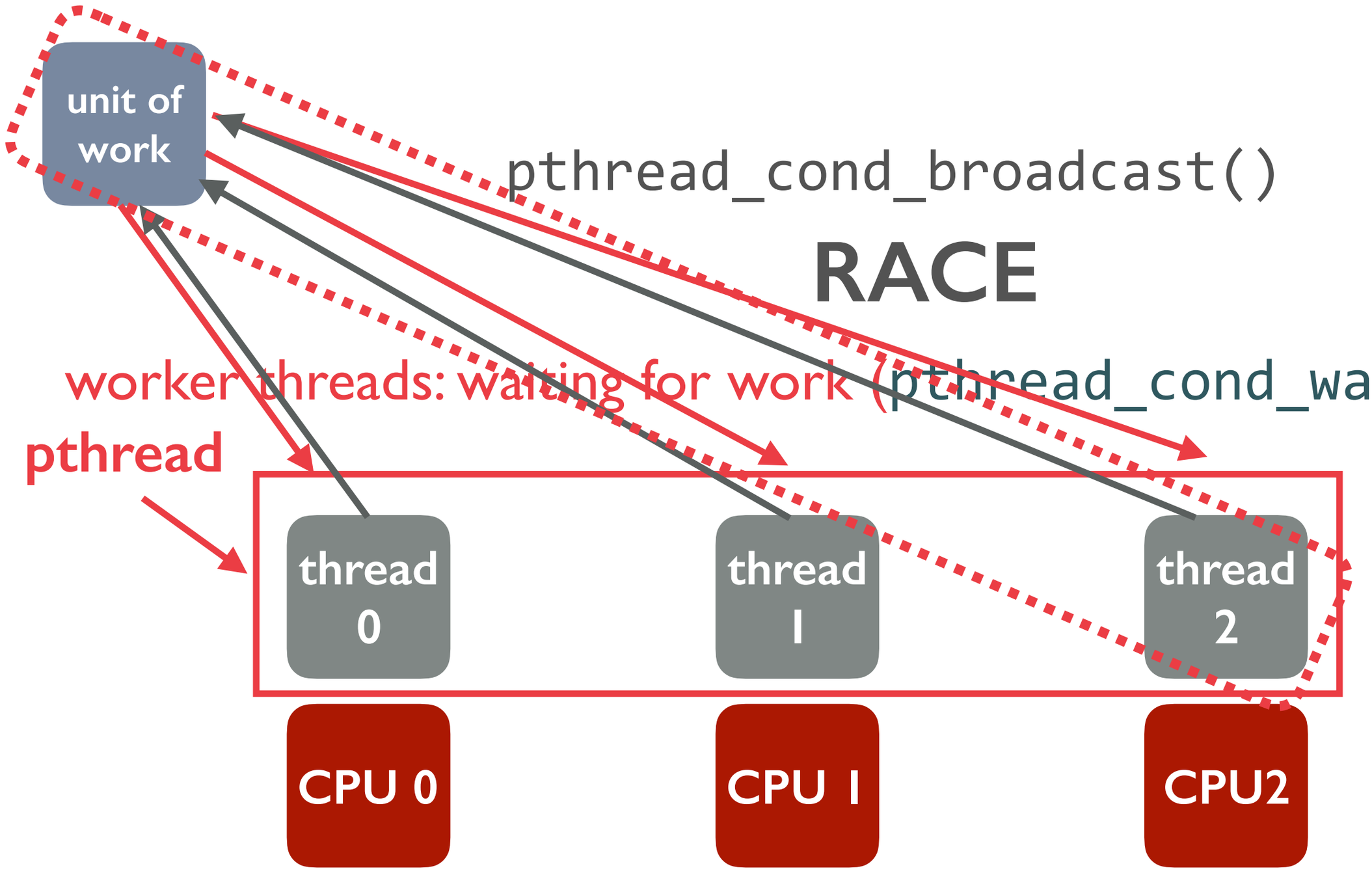
message arrival

work is completed

work is available

something terrible happened

AN EXAMPLE: LEGION

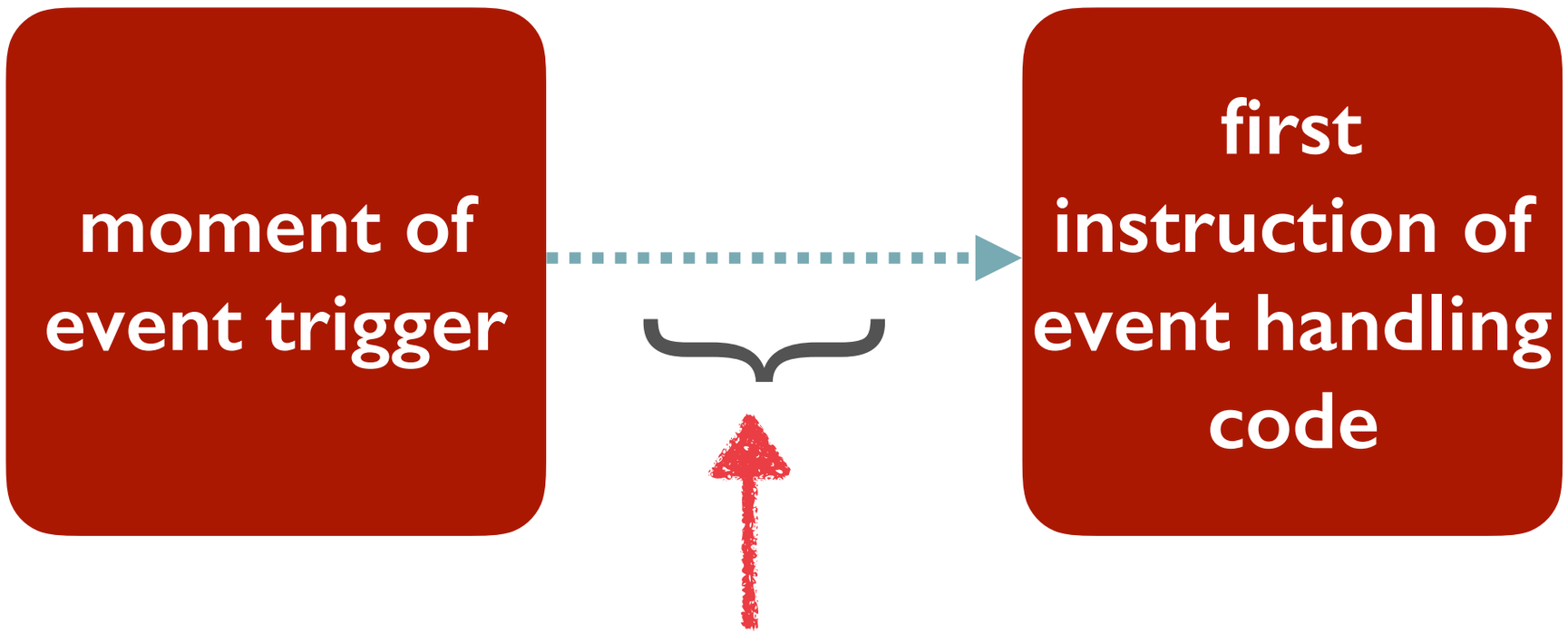


ASYNCHRONOUS EVENTS

the receiving side is **not blocked**

other things can run

WE WANT **FAST** EVENTS



we want to minimize this

WHAT'S THE LOWER LIMIT?

light!

what we want: **SoL**†

what we actually get with existing software events: **SoL**††

†speed of light

††s**t out of luck

OUTLINE

software abstractions for asynchronous events

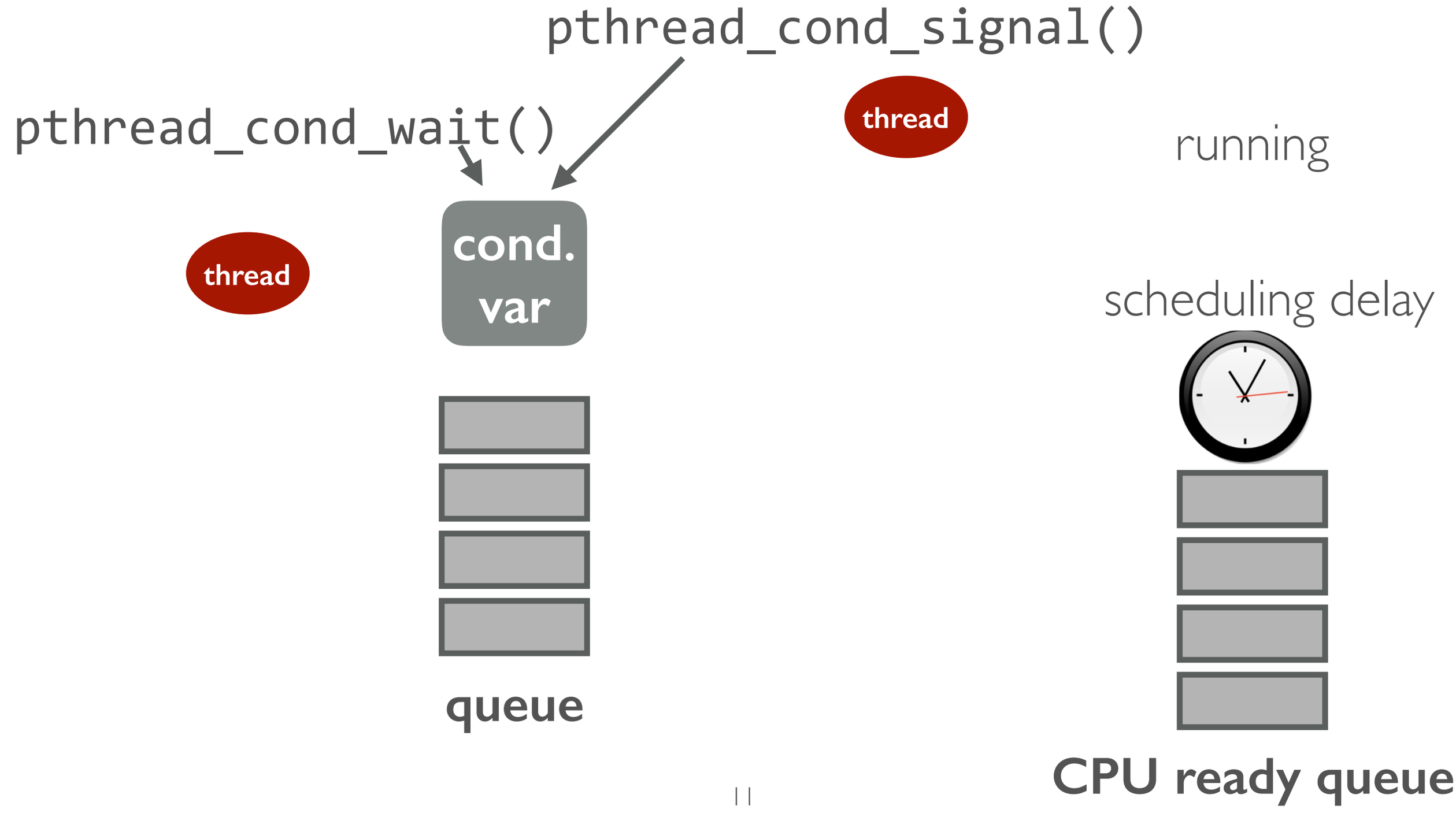
hardware capabilities

event performance

NEMO: benefits of kernel mode

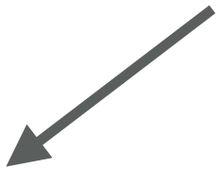
NEMO: closer to the hardware

CONDITION VARIABLES



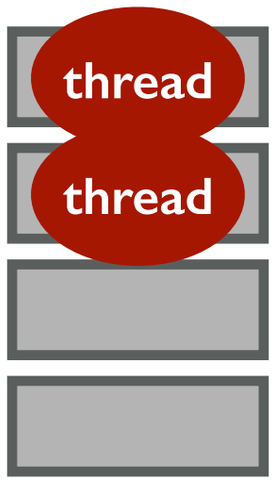
BROADCAST

```
pthread_cond_broadcast()
```

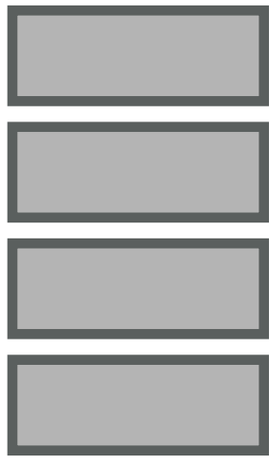


thread

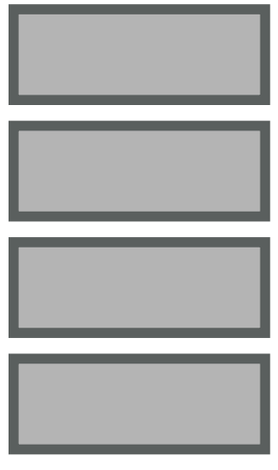
cond. var



queue



CPU 0
ready queue



CPU 1
ready queue

IMMEDIATELY VISIBLE ISSUES

we're at the behest of the scheduler

broadcast is linear in number of waiters

we can't tell scheduler to initiate a "fast" wakeup

OUTLINE

software abstractions for asynchronous events

hardware capabilities

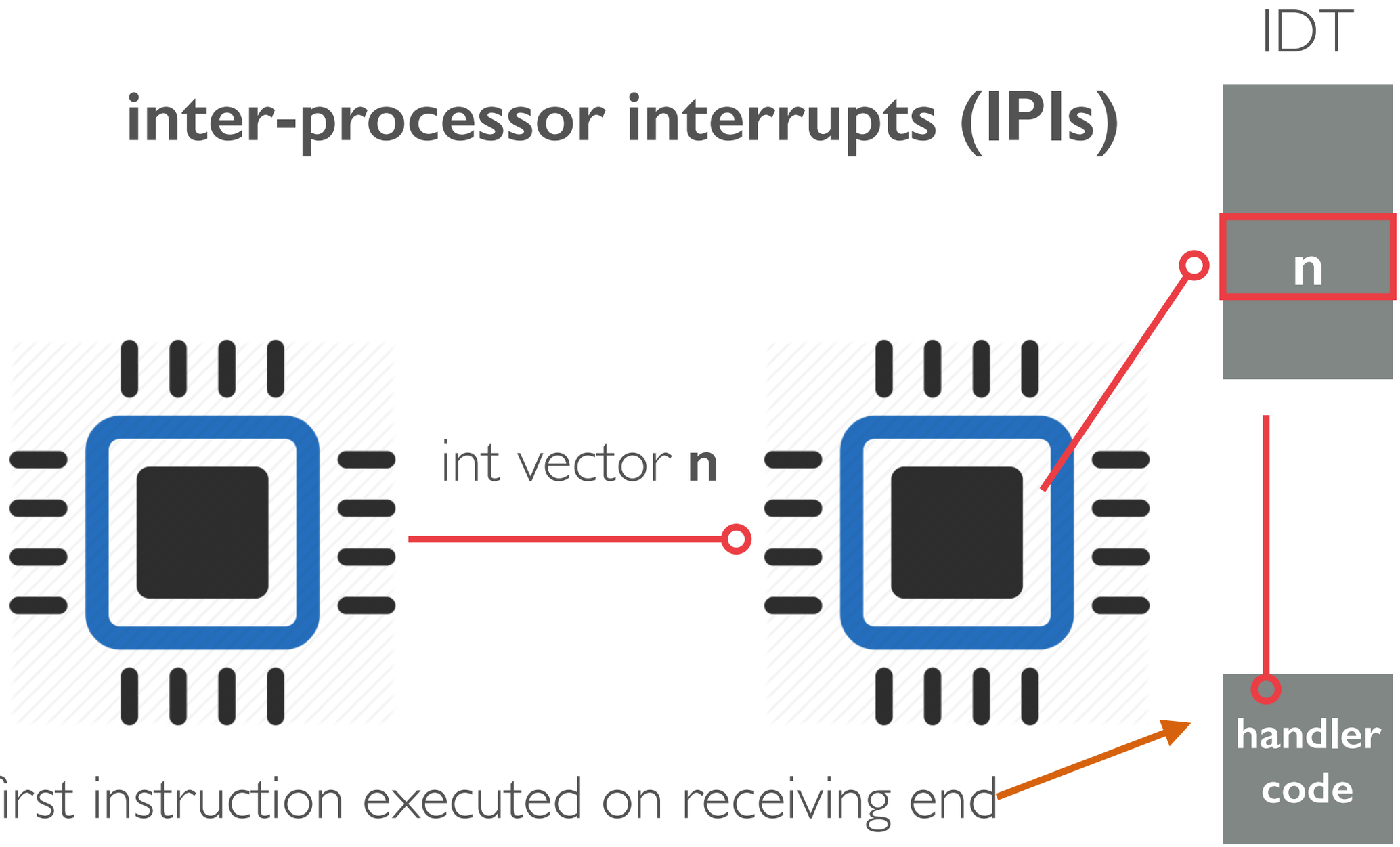
event performance

NEMO: benefits of kernel mode

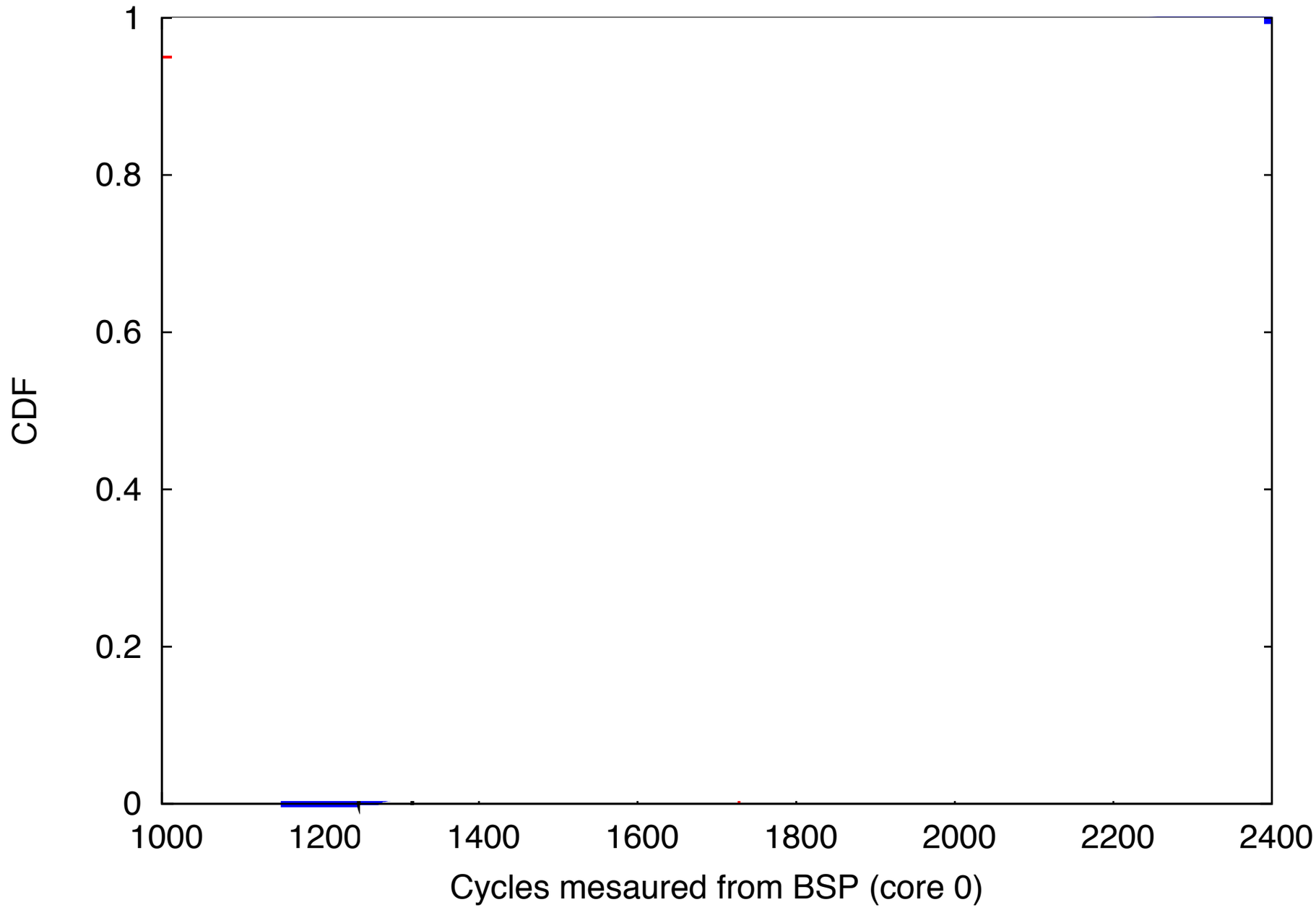
NEMO: closer to the hardware

WHAT CAN WE DO IN HARDWARE?

inter-processor interrupts (IPIs)



IPIS ARE *FAST*



OUTLINE

software abstractions for asynchronous events

hardware capabilities

event performance

NEMO: benefits of kernel mode

NEMO: closer to the hardware

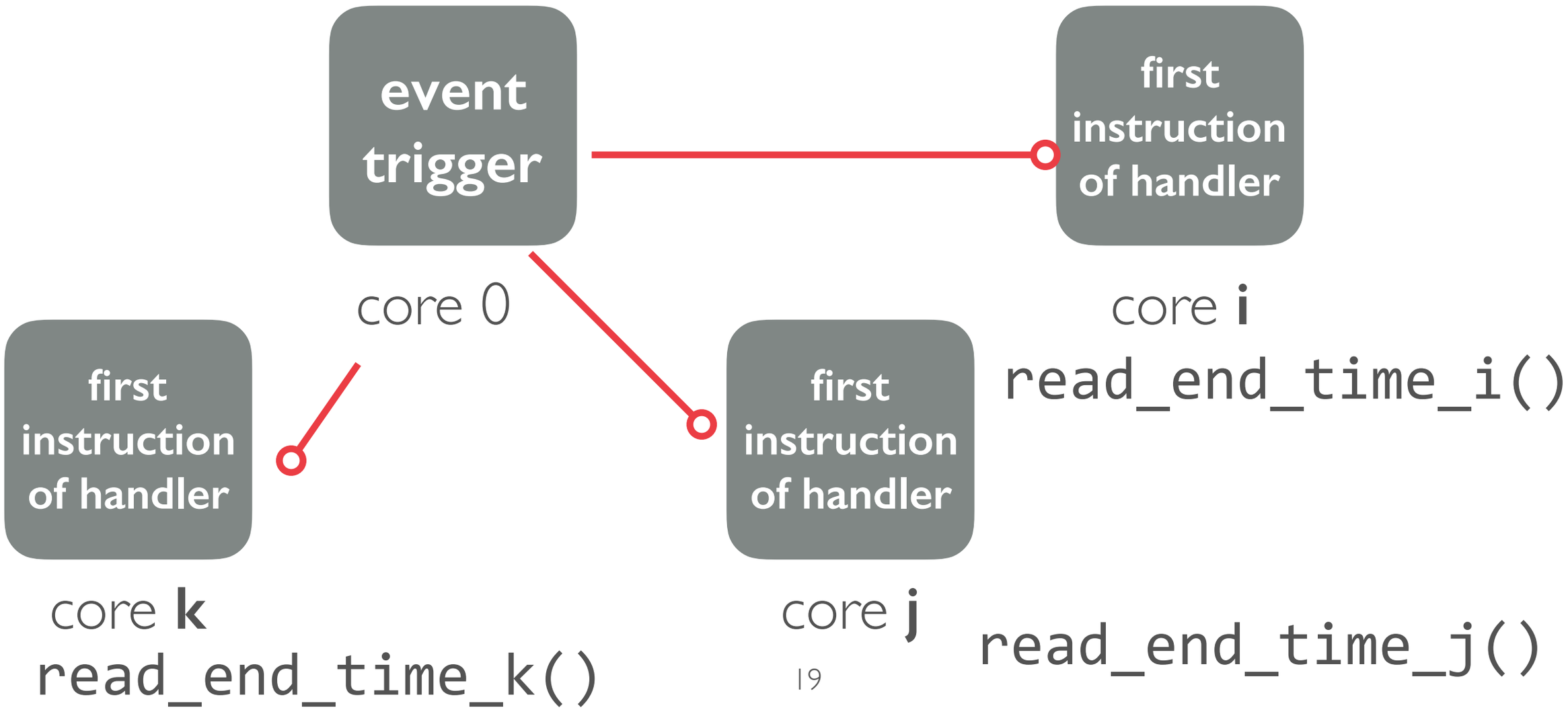
MEASURING EVENT WAKEUP LATENCY

`read_start_time()`

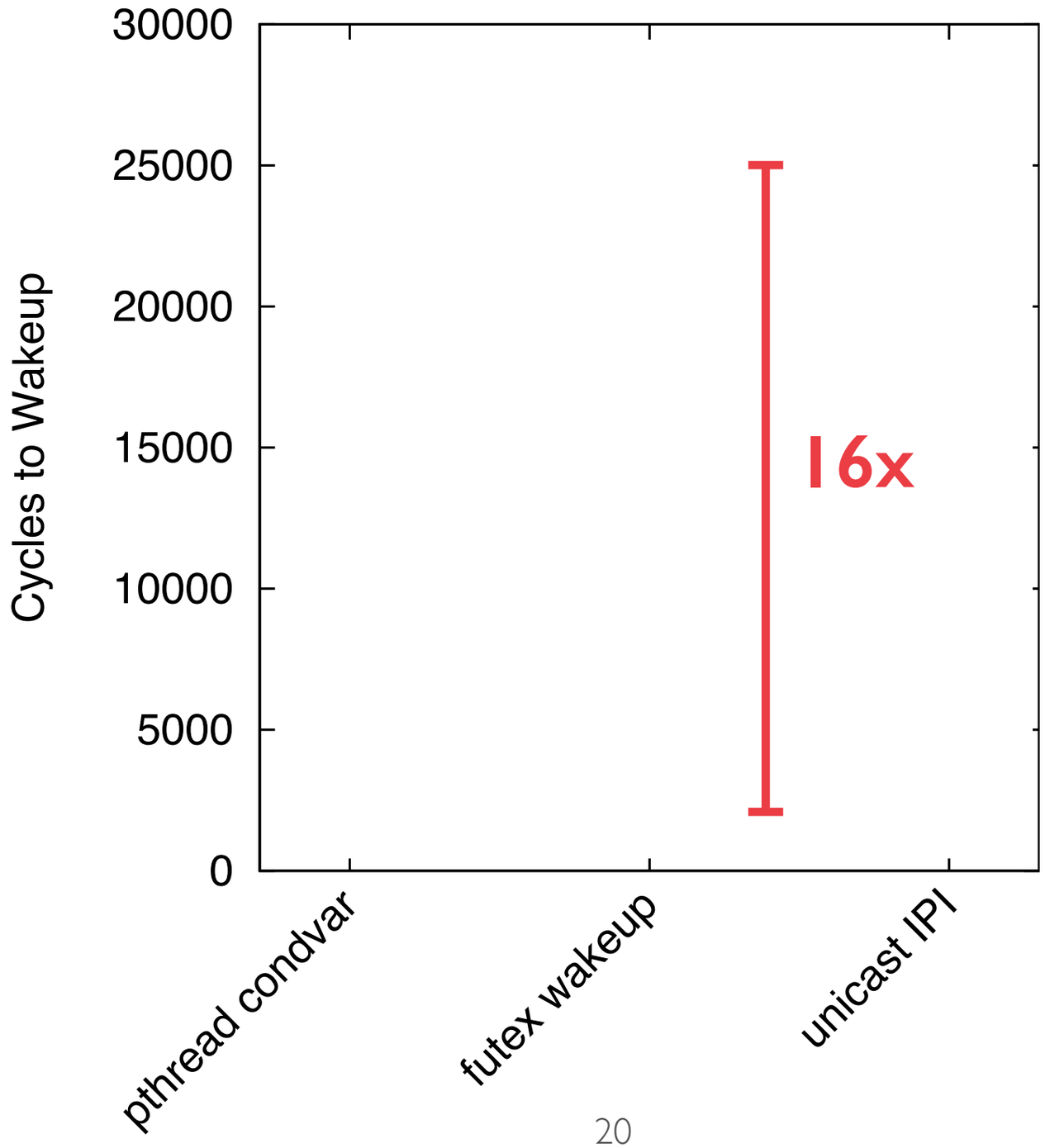
`read_end_time()`

MEASURING EVENT BROADCAST LATENCY

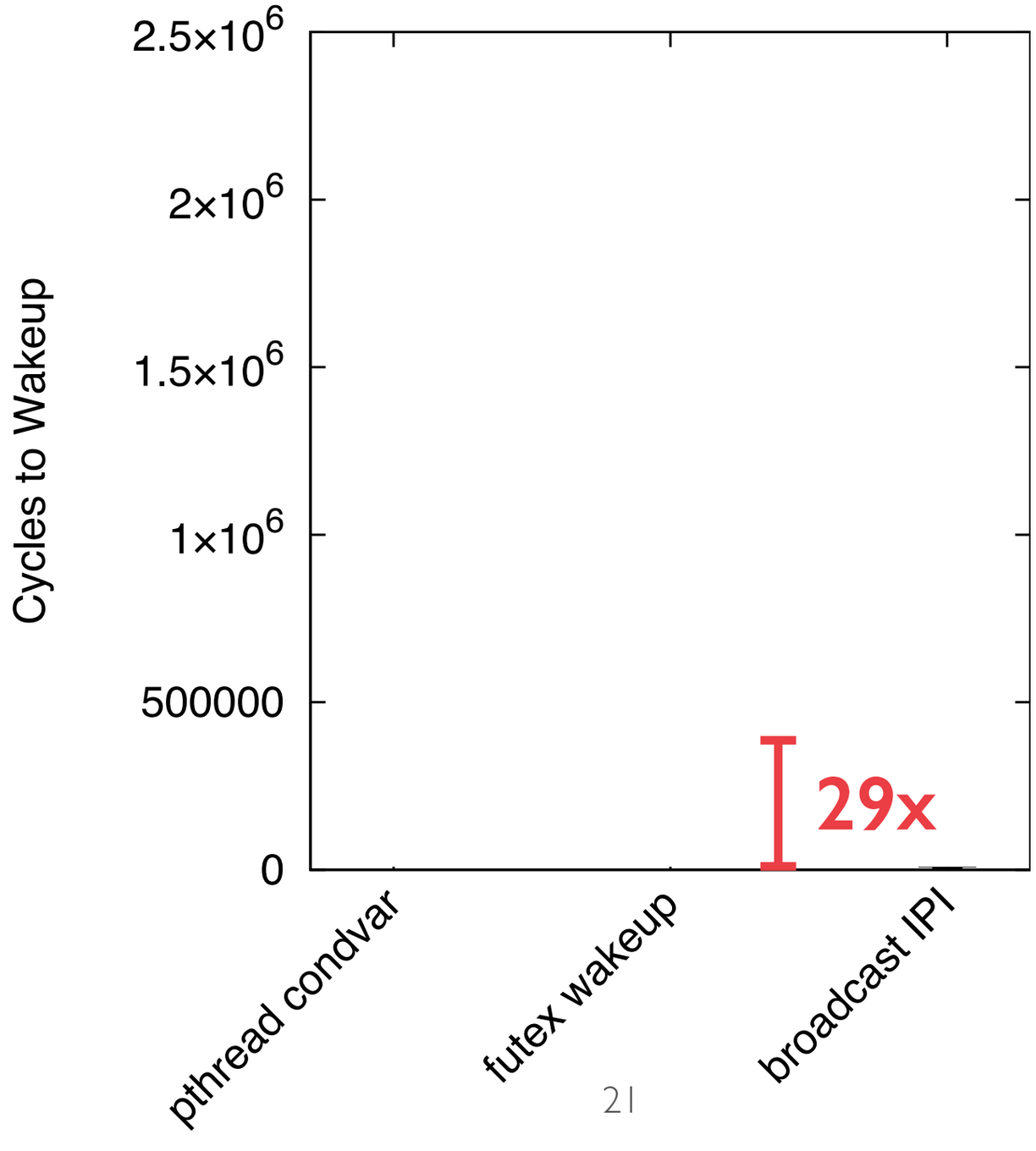
`read_start_time()`



EXISTING SOFTWARE EVENTS ARE **SLOW**



BROADCASTS ARE ALSO TERRIBLE



SYNCHRONY

for broadcasts, we want events to be delivered to all cores at the same time

useful for, e.g. BSP apps with events

measure the *deviation* of wakeup time across cores in a broadcast

SYNCHRONY

70x difference between hardware IPIs and software mechanisms

hardly any predictability!

OUTLINE

software abstractions for asynchronous events

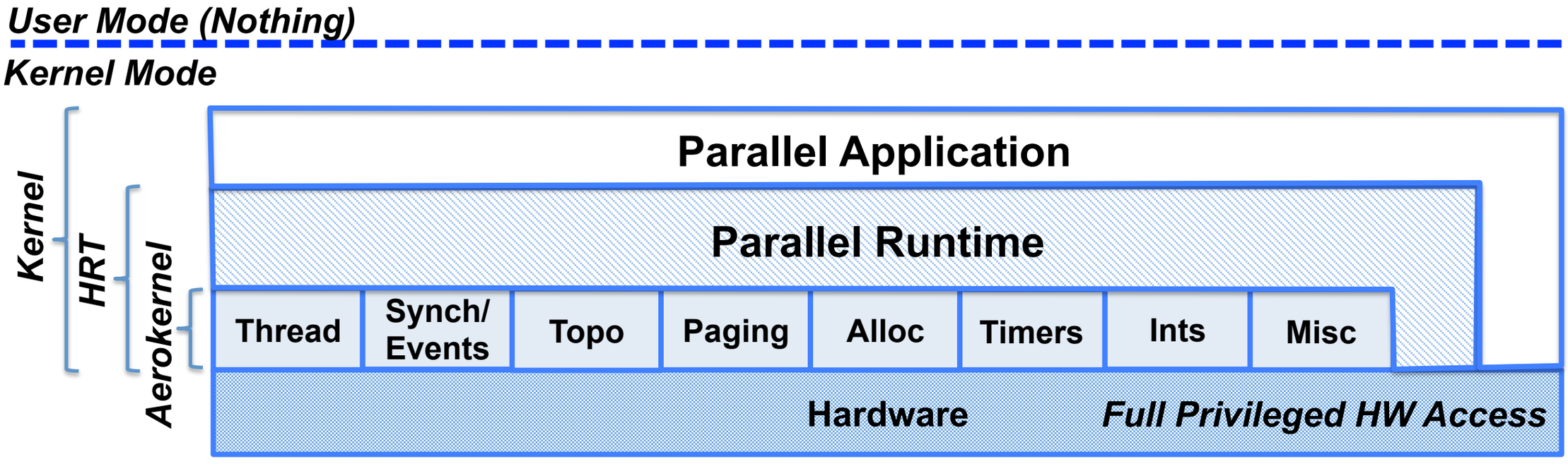
hardware capabilities

event performance

NEMO: benefits of kernel mode

NEMO: closer to the hardware

NAUTILUS



[Hale, Dinda HPDC '15]

[Hale, Dinda VEE '16]

[Hale, Hetland, Dinda FRIDAY]


nemo
accelerated, asynchronous software events

RETAINING FAMILIAR INTERFACES

use a lightweight, kernel-mode framework (like Nautilus) to eliminate overheads

maintain userspace interfaces (e.g. condition variable wait,signal,broadcast etc.)

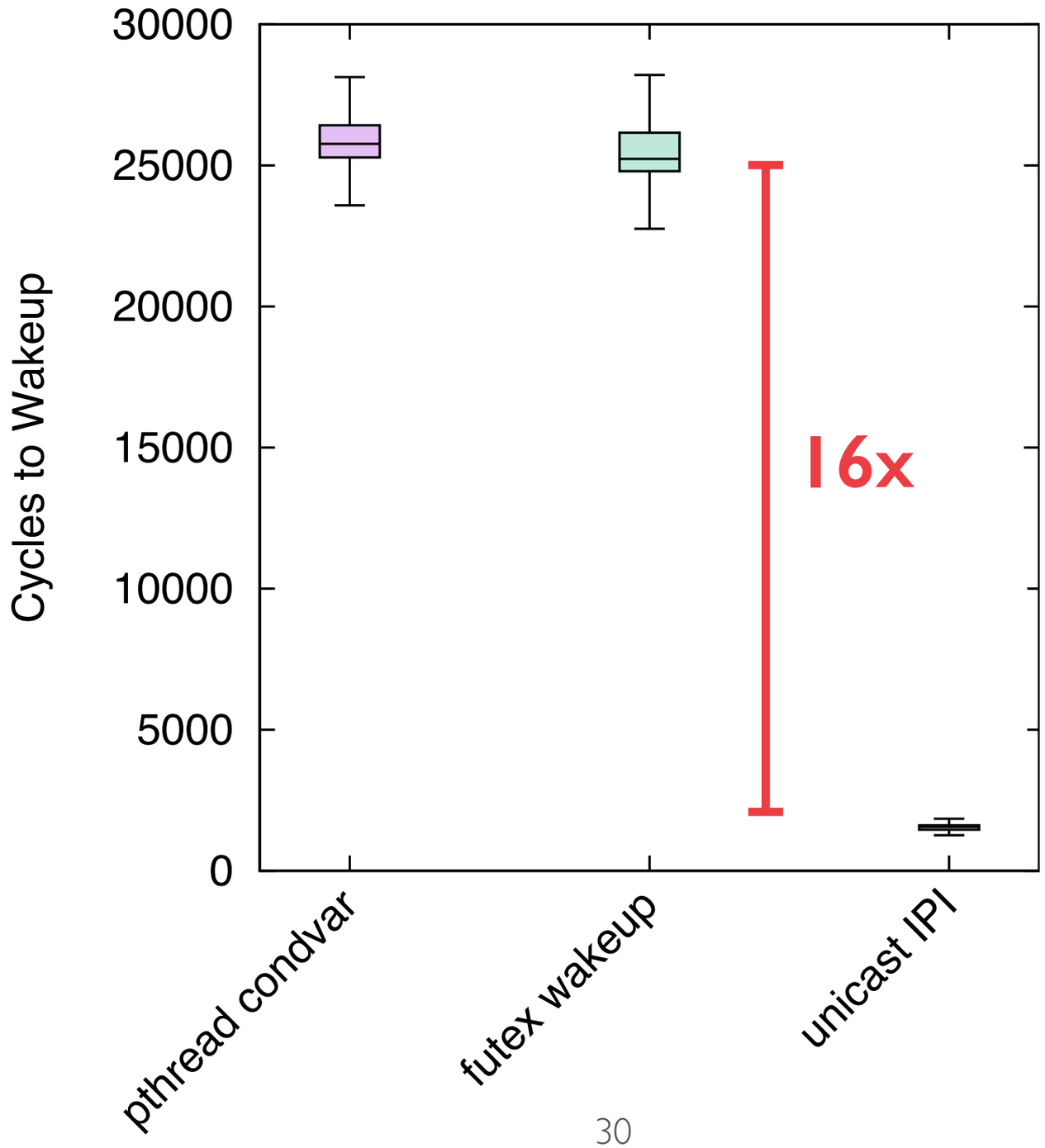
if we build our kernel from scratch, how fast can we get?

NEMO HAS 2 COMPATIBLE CONDITION VARIABLE IMPLEMENTATIONS

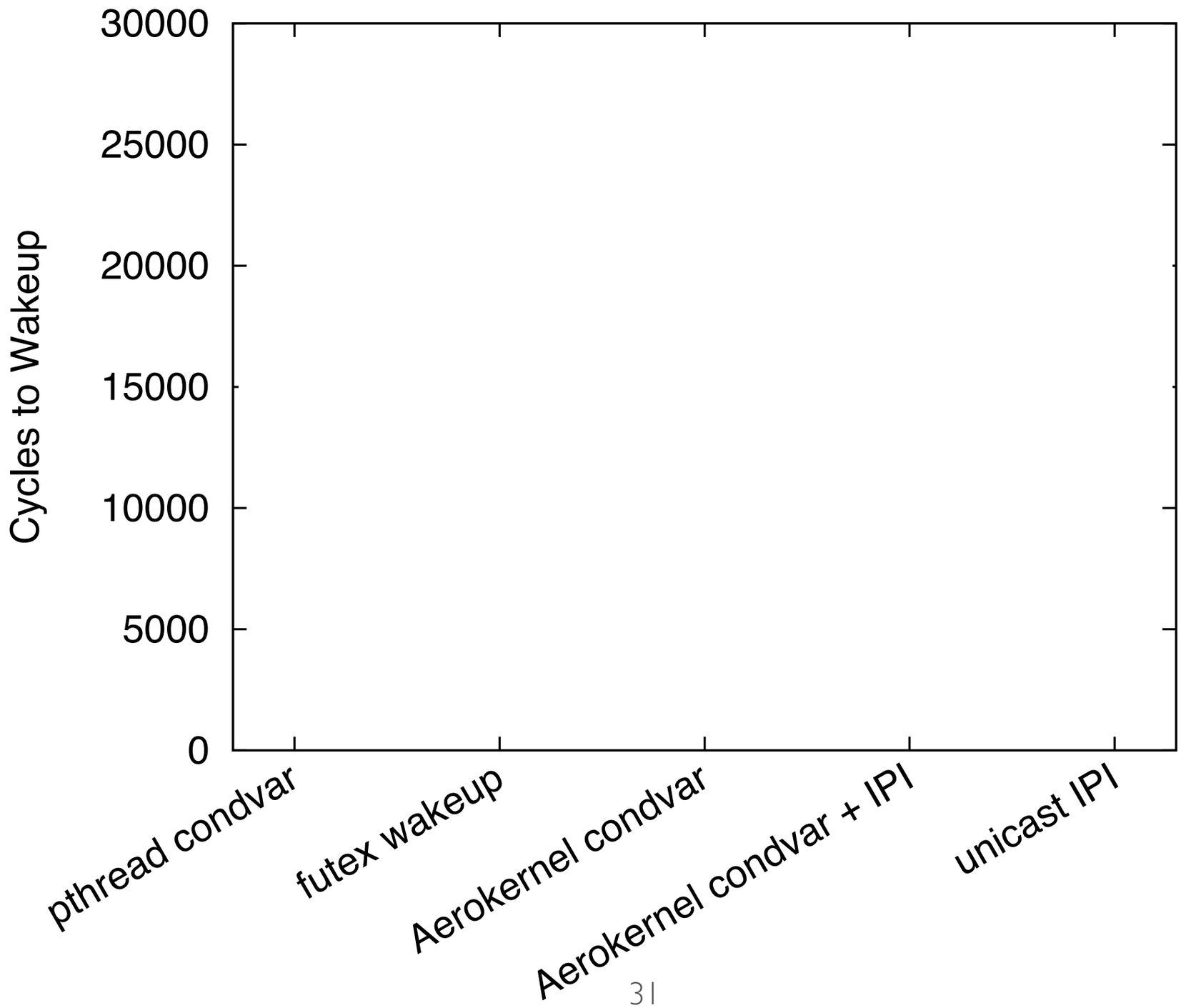
lightweight condition variables

leverage IPI access to “kick” the scheduler

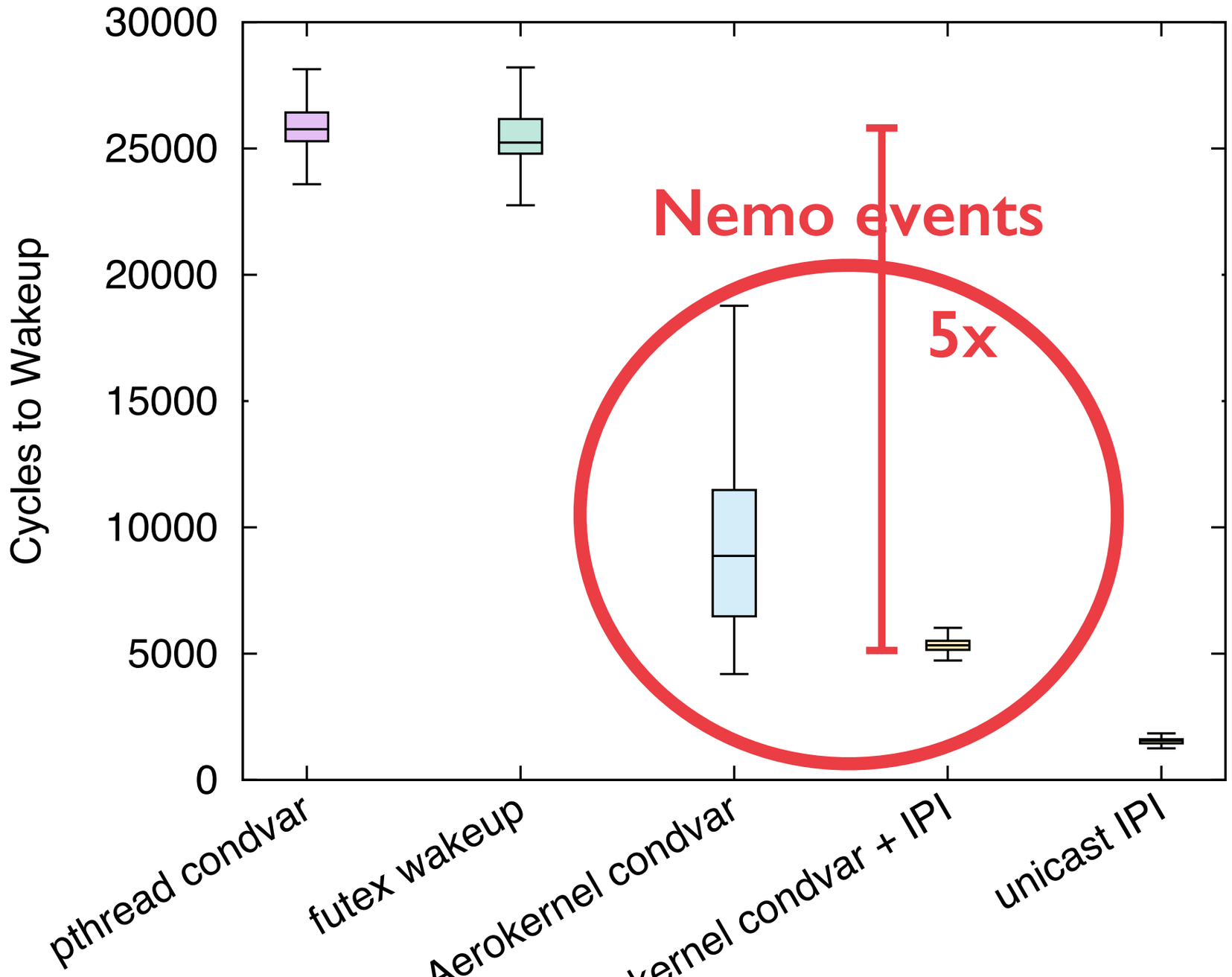
EXISTING SOFTWARE EVENTS ARE **SLOW**



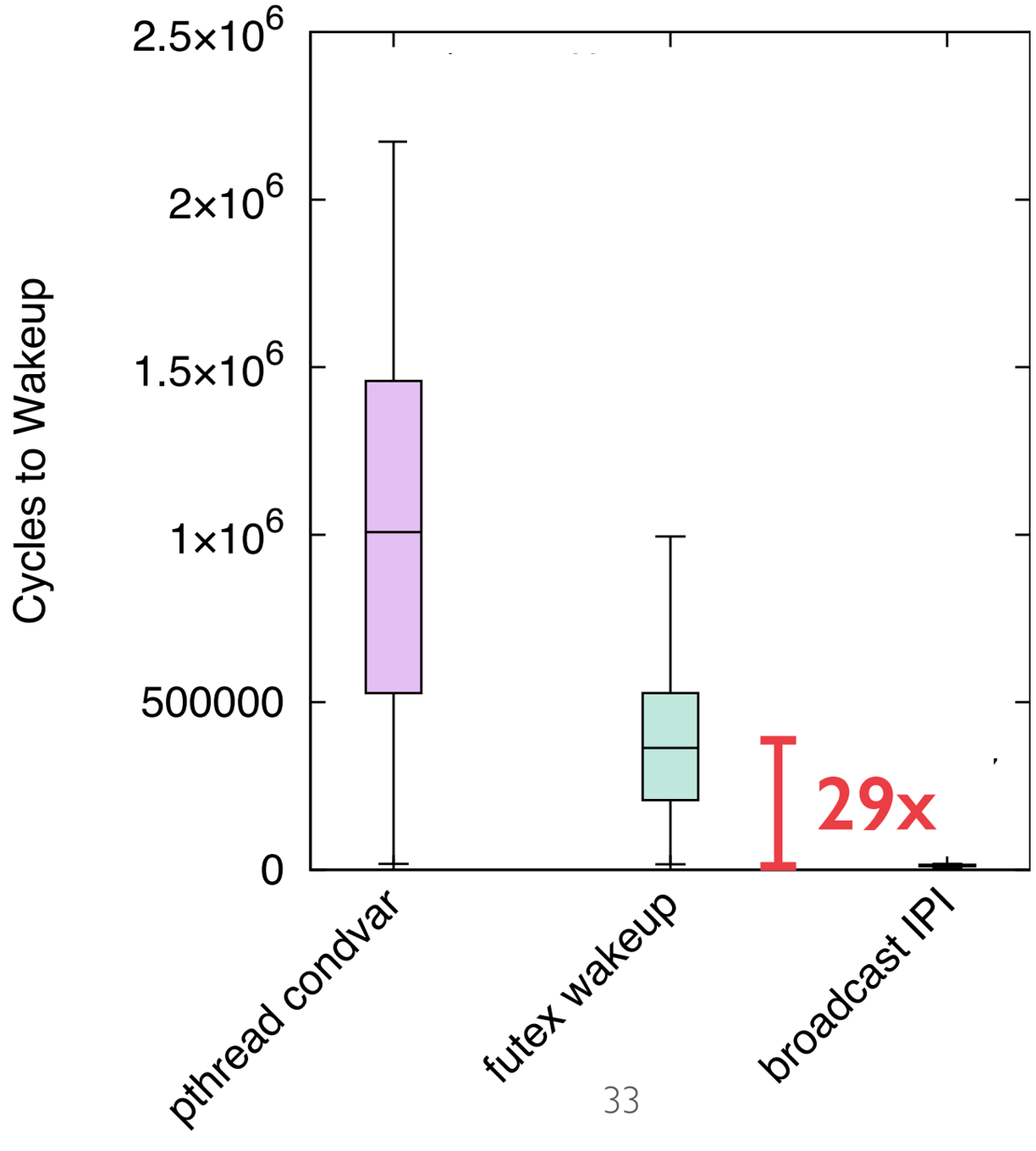
NEMO SPEEDS THINGS UP



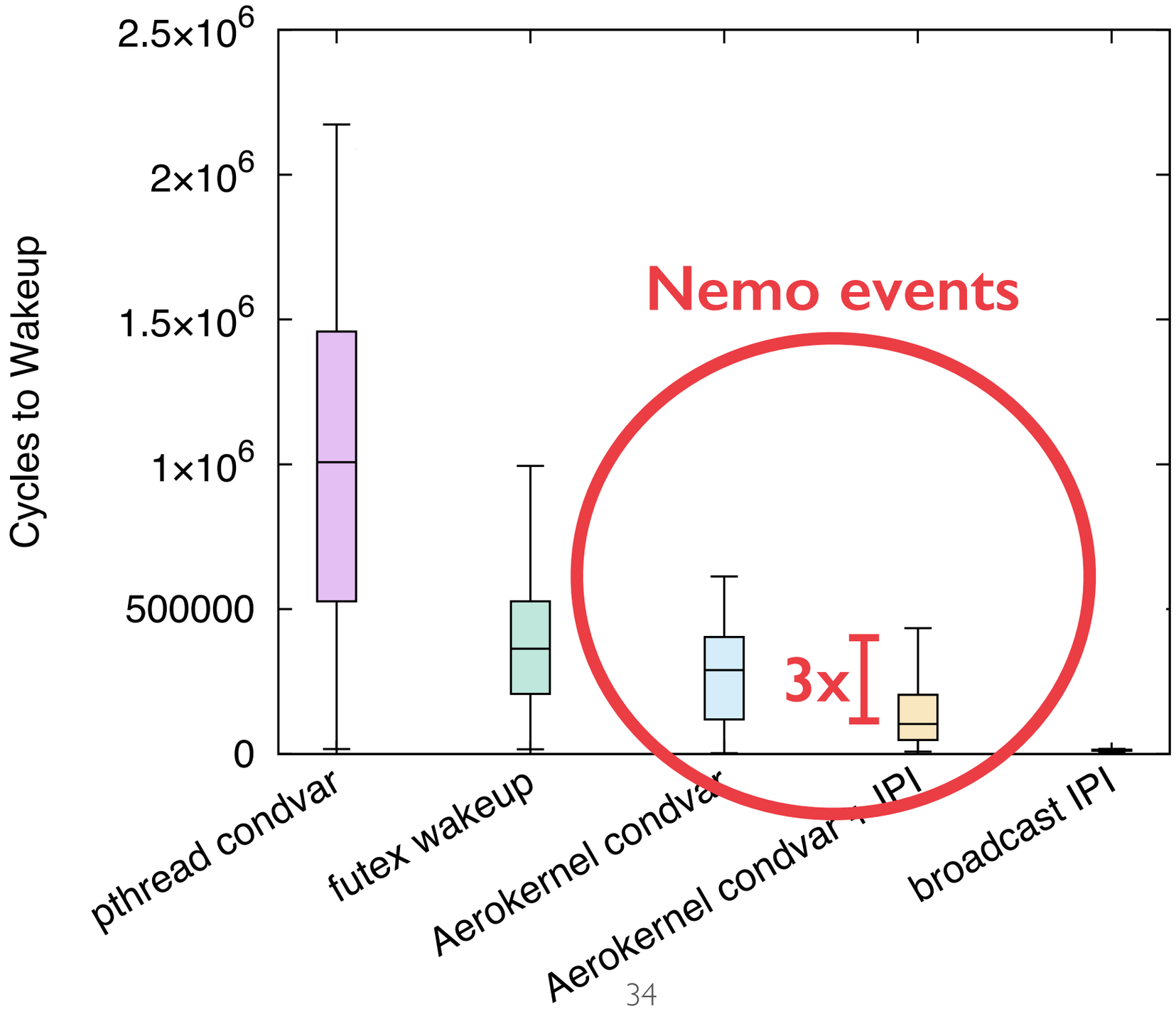
NEMO SPEEDS THINGS UP



BROADCASTS ARE ALSO TERRIBLE



NEMO BRINGS US CLOSER TO IPI BROADCAST LATENCY



SYNCHRONY

we can do 2x better than user-space mechanisms

(with compatible interfaces)

OUTLINE

software abstractions for asynchronous events

hardware capabilities

event performance

NEMO: benefits of kernel mode

NEMO: closer to the hardware

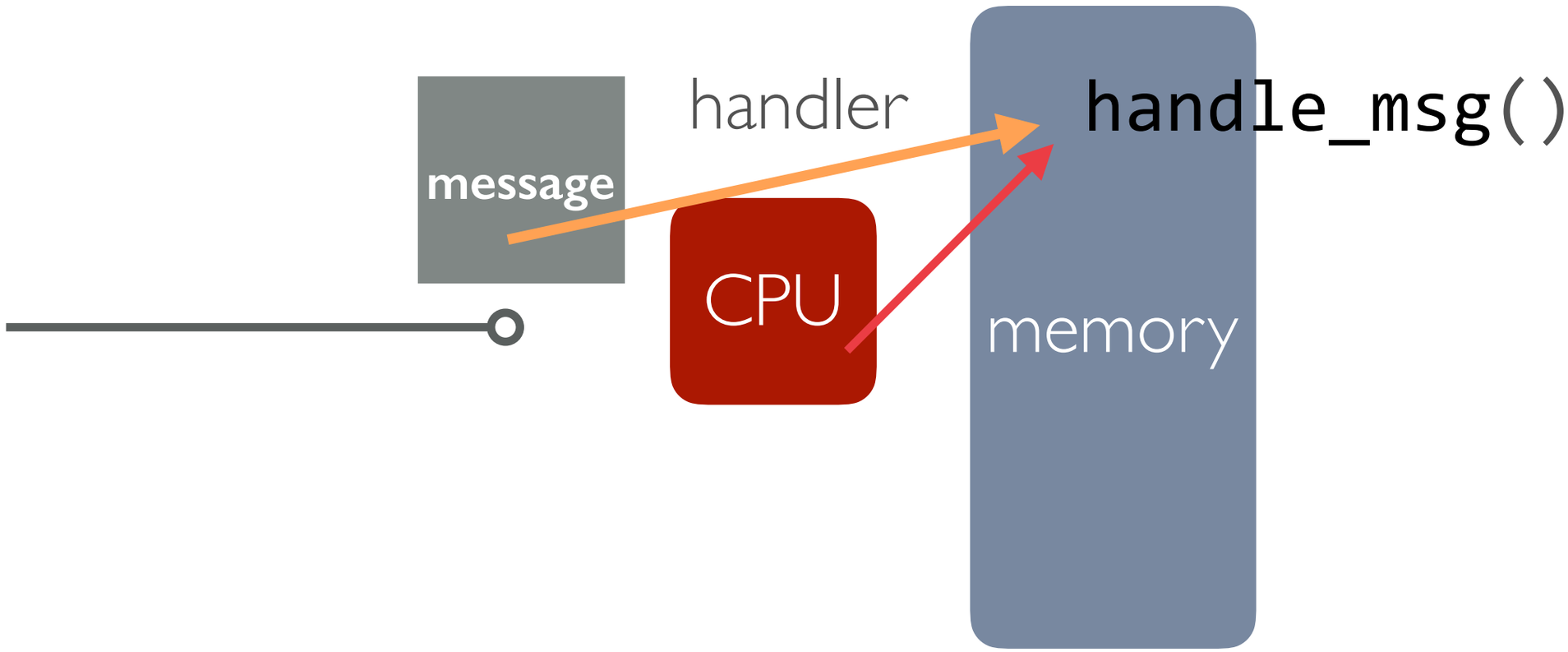
WHAT IF WE GIVE UP THE FAMILIAR INTERFACE?

modify condition variable semantics

we don't necessarily care which context (thread) receives the event, as long as it's handled at a particular core

not appropriate for all situations

ACTIVE MESSAGES



claim: better fit than, e.g. cond vars, for many event-based schemes

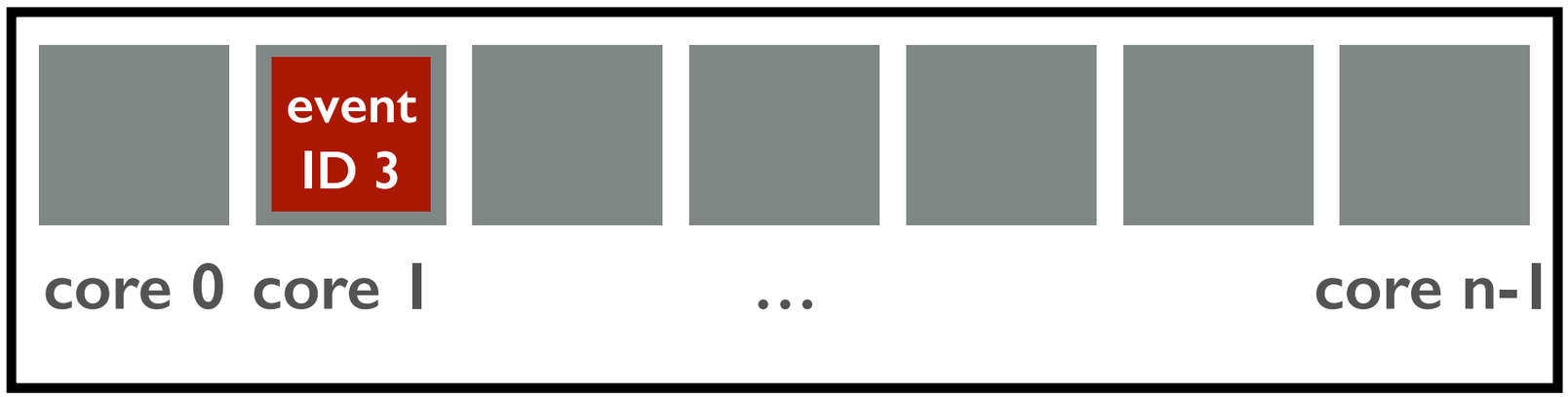
we want to use IPIs as an active message substrate

problem: IPIs don't have a payload!

allocate several **event IDs**

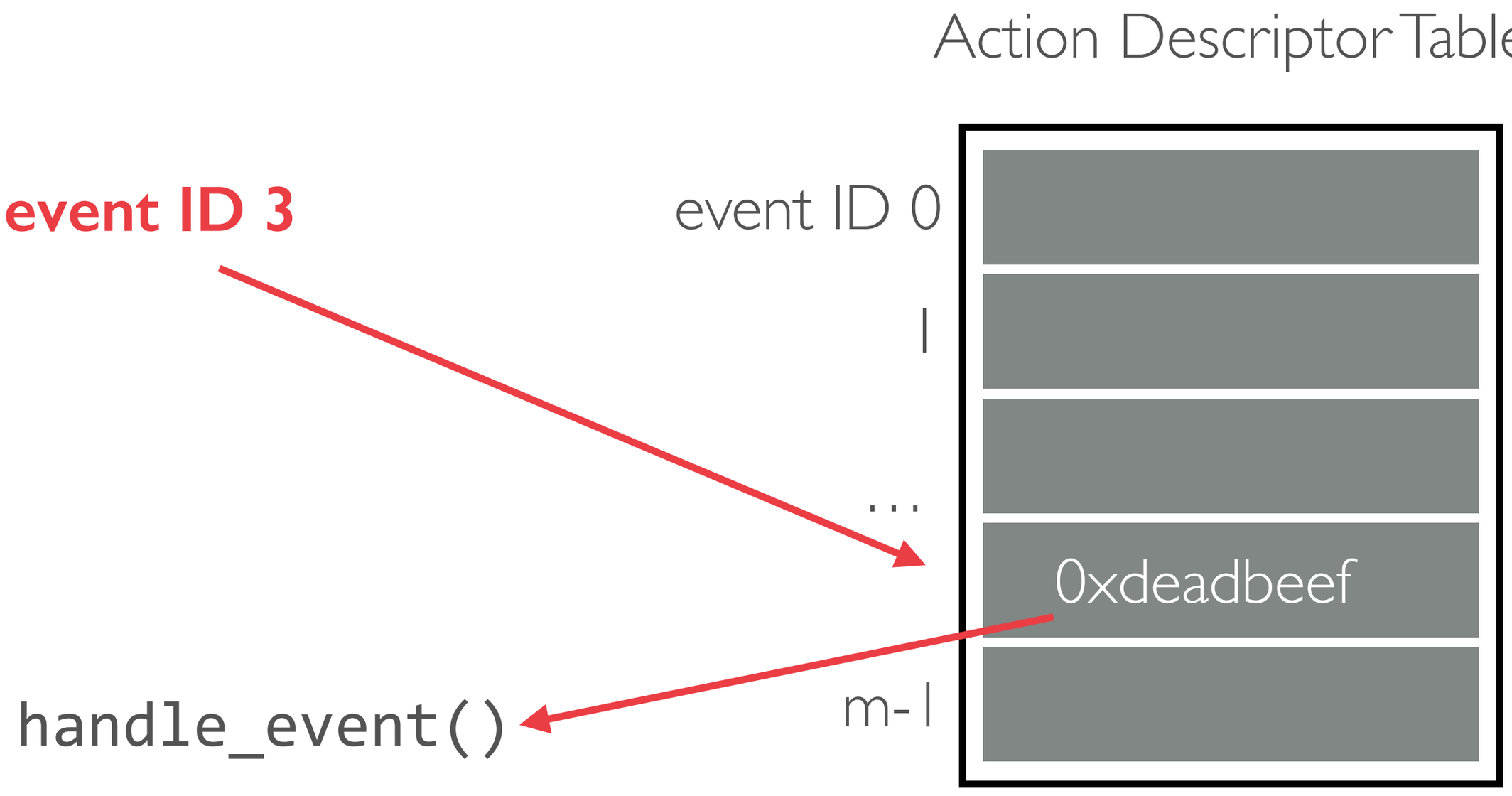
when core receives interrupt,
lookup the event ID in a table indexed on core ID

Action Lookup Table

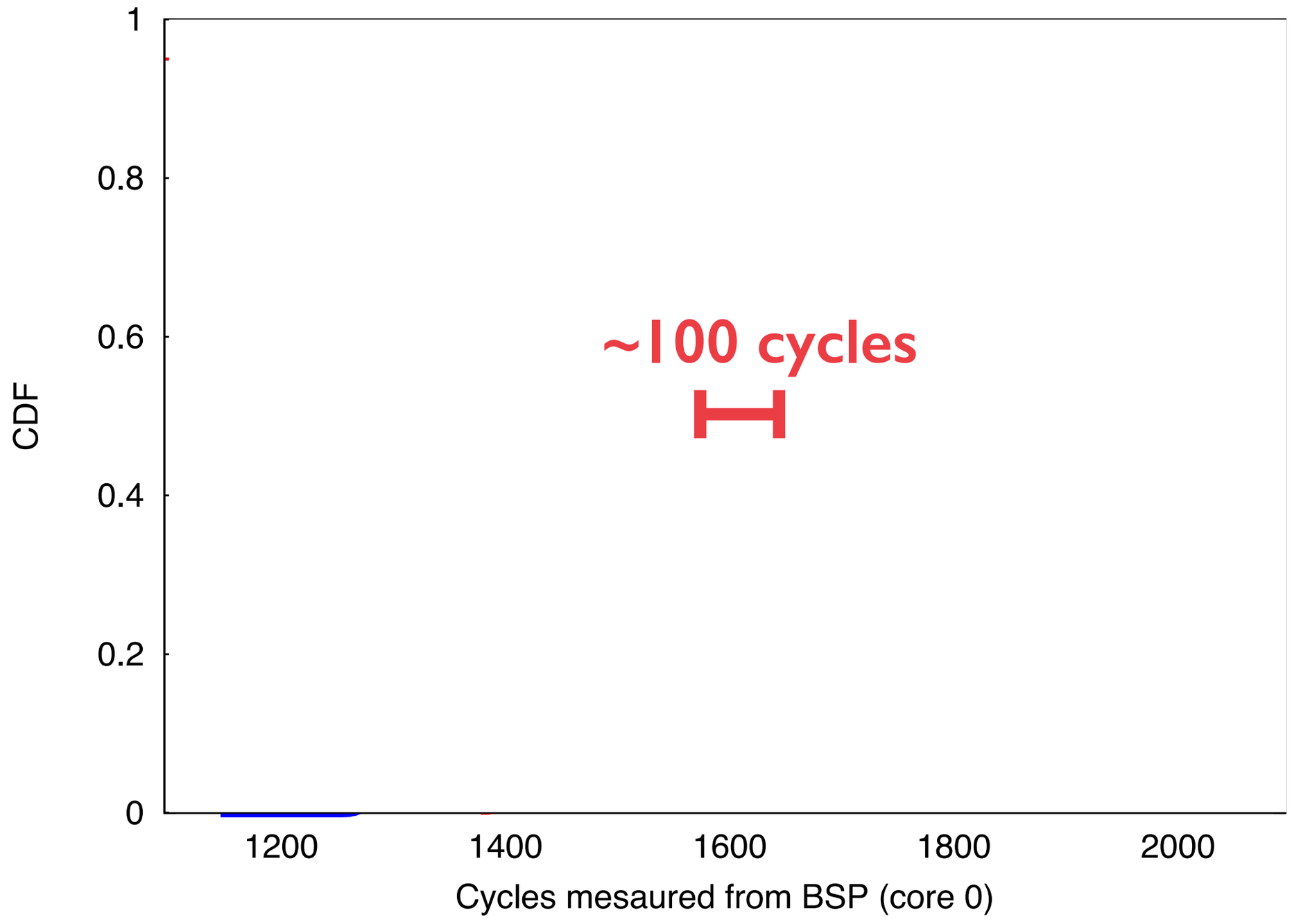



```
nemo_notify_event(core=1, event=3)
```

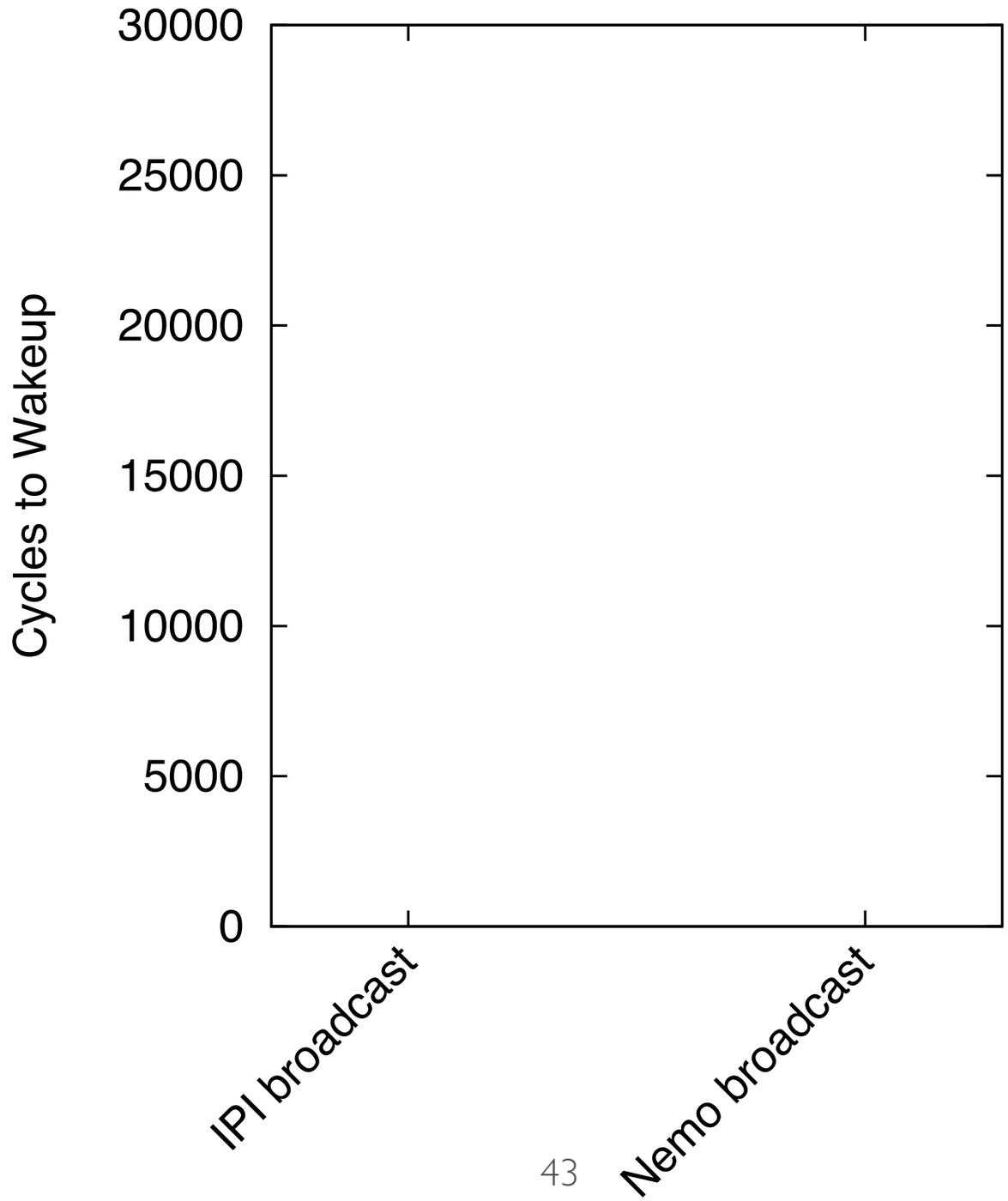
event ID corresponds to an "action" (a handler)



NEMO WAKEUPS HAVE CONSTANT OFFSET FROM IPIS



BROADCAST LATENCY ALSO ON PAR WITH IPIS



NEMO ACHIEVES TIGHT SYNCHRONY

< 50 cycles variation in broadcast wakeups between cores

SUMMARY

if you want asynch. event delivery close to hardware latency…

existing mechanisms are pretty terrible

SOME WAYS TO FIX IT:

throw out general purpose OS abstractions (e.g. user/kernel boundary)

throw out typical event abstractions

use the hardware directly!

THANKS

ありがとう

http://halek.co ← **me**

http://presciencelab.org ← **our lab**

http://nautilus.halek.co ← **Nautilus**

http://xstack.sandia.gov/hobbes

Hobbes Exascale OS/R project

Northwestern University

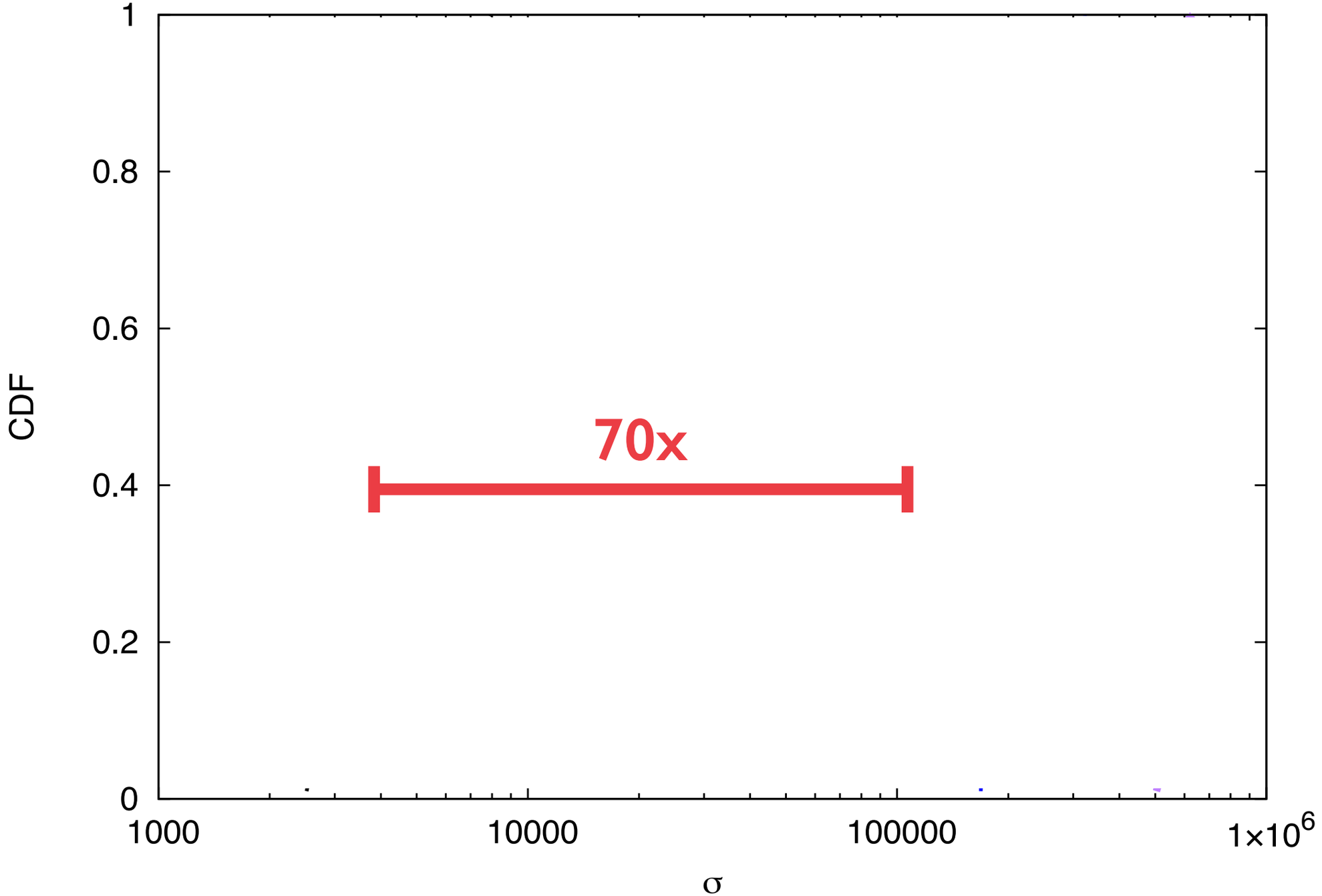
HOBBES

xstack.sandia.gov/hobbes

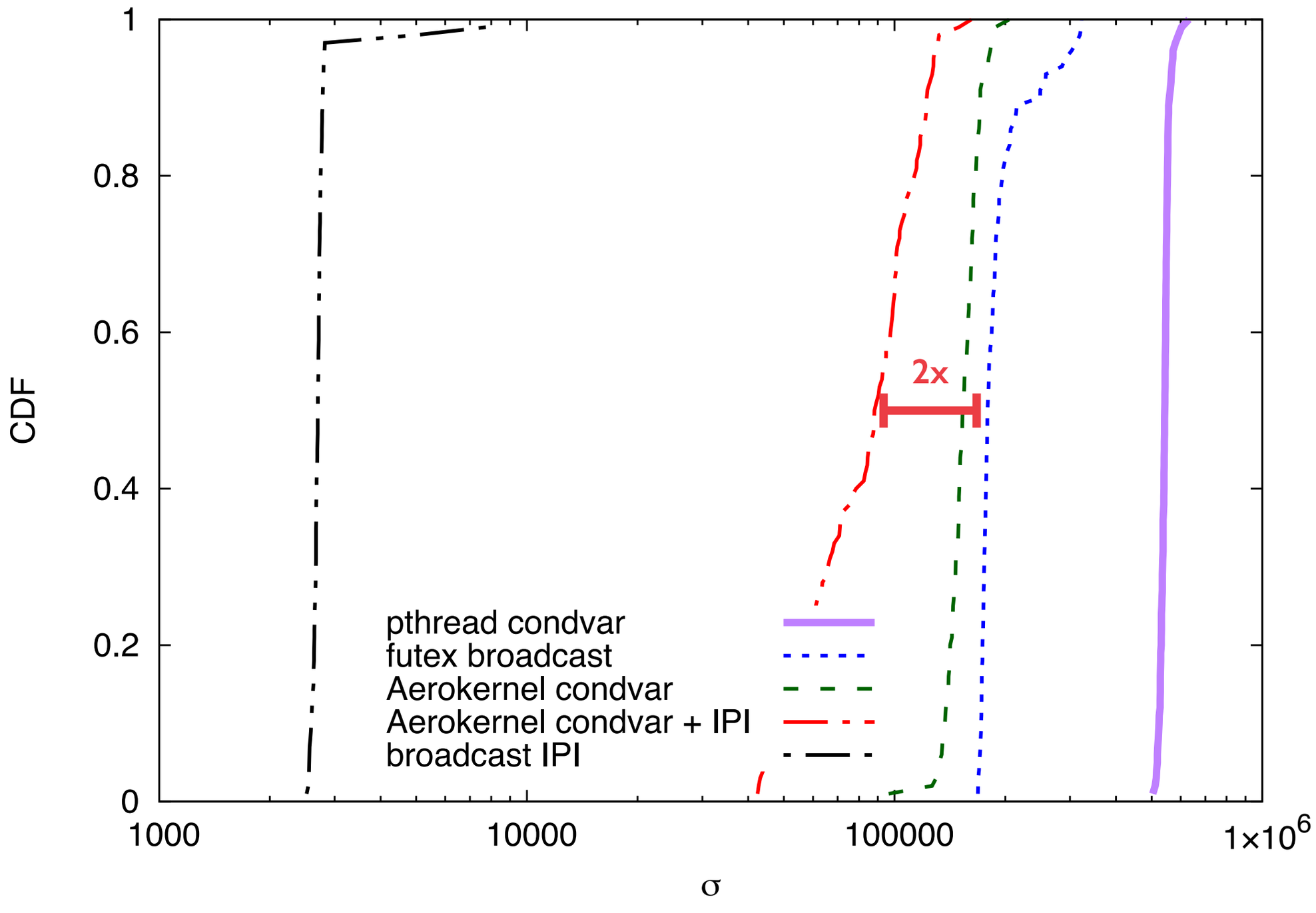
ILLINOIS INSTITUTE OF TECHNOLOGY

BACKUPS

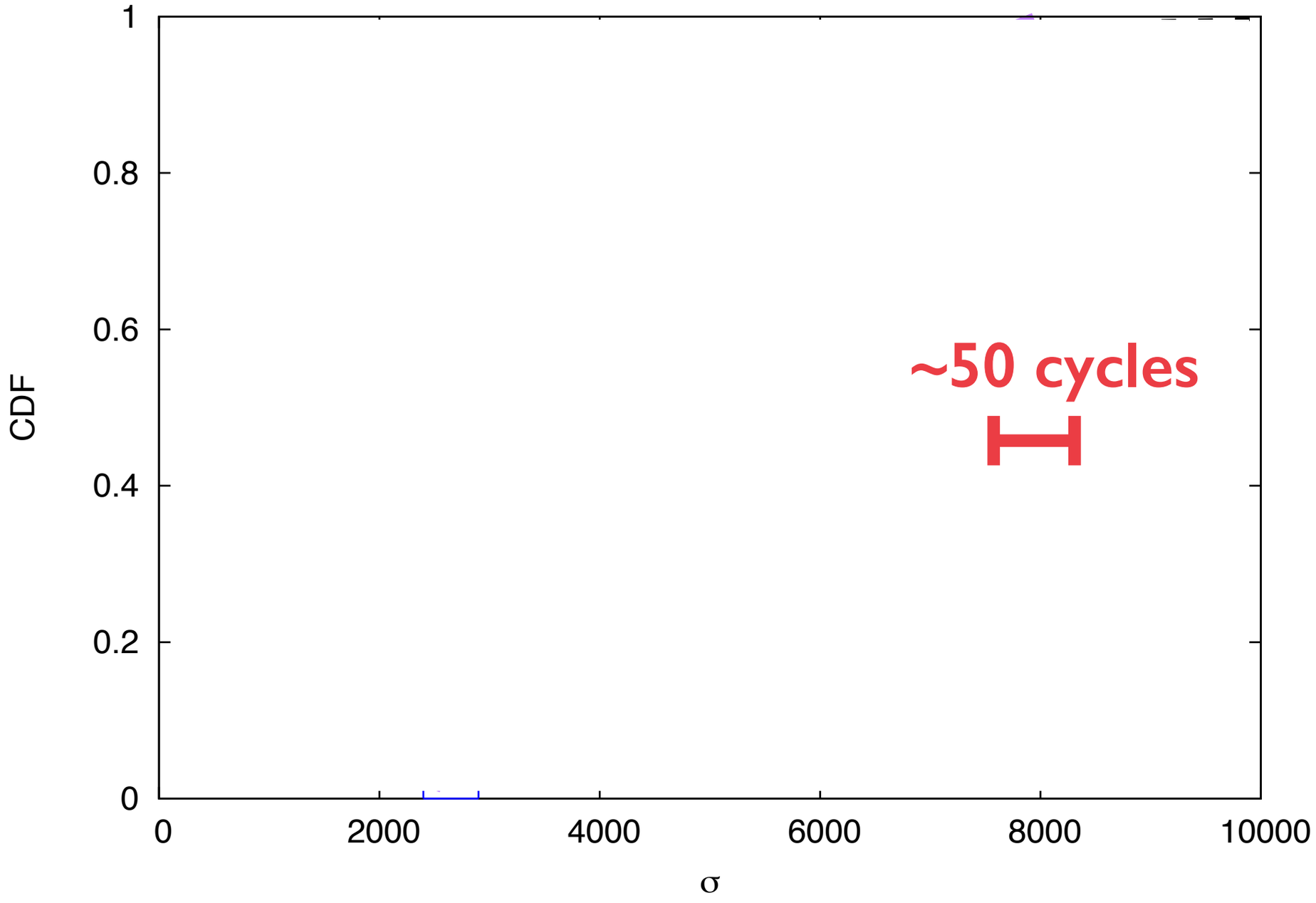
TIGHT SYNCHRONY FOR IPIS, NOT FOR SOFTWARE EVENTS



NEMO GETS US CLOSER



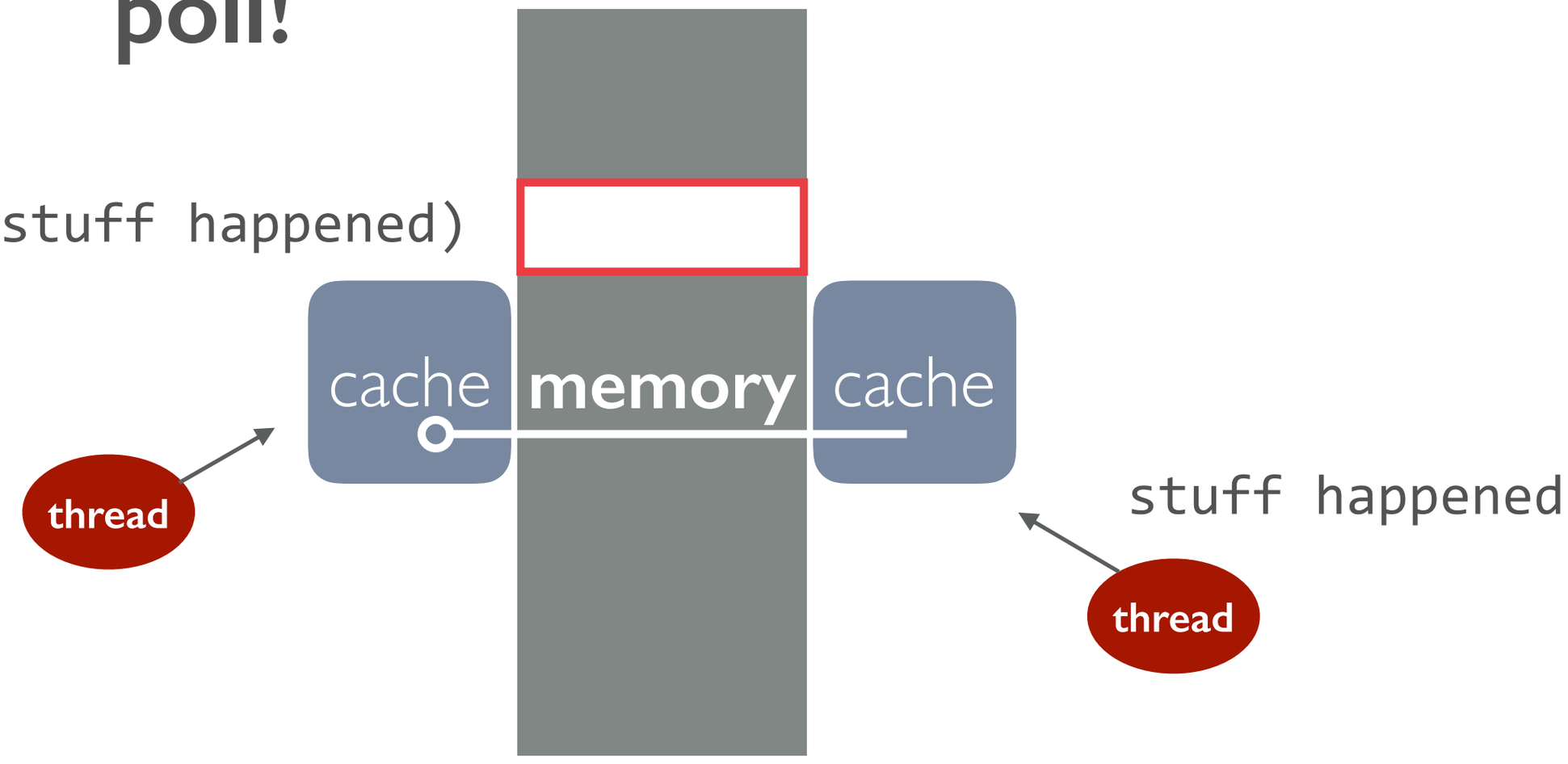
NEMO ACHIEVES TIGHT SYNCHRONY



HOW FAST CAN A NOTIFICATION BE IN H/W?

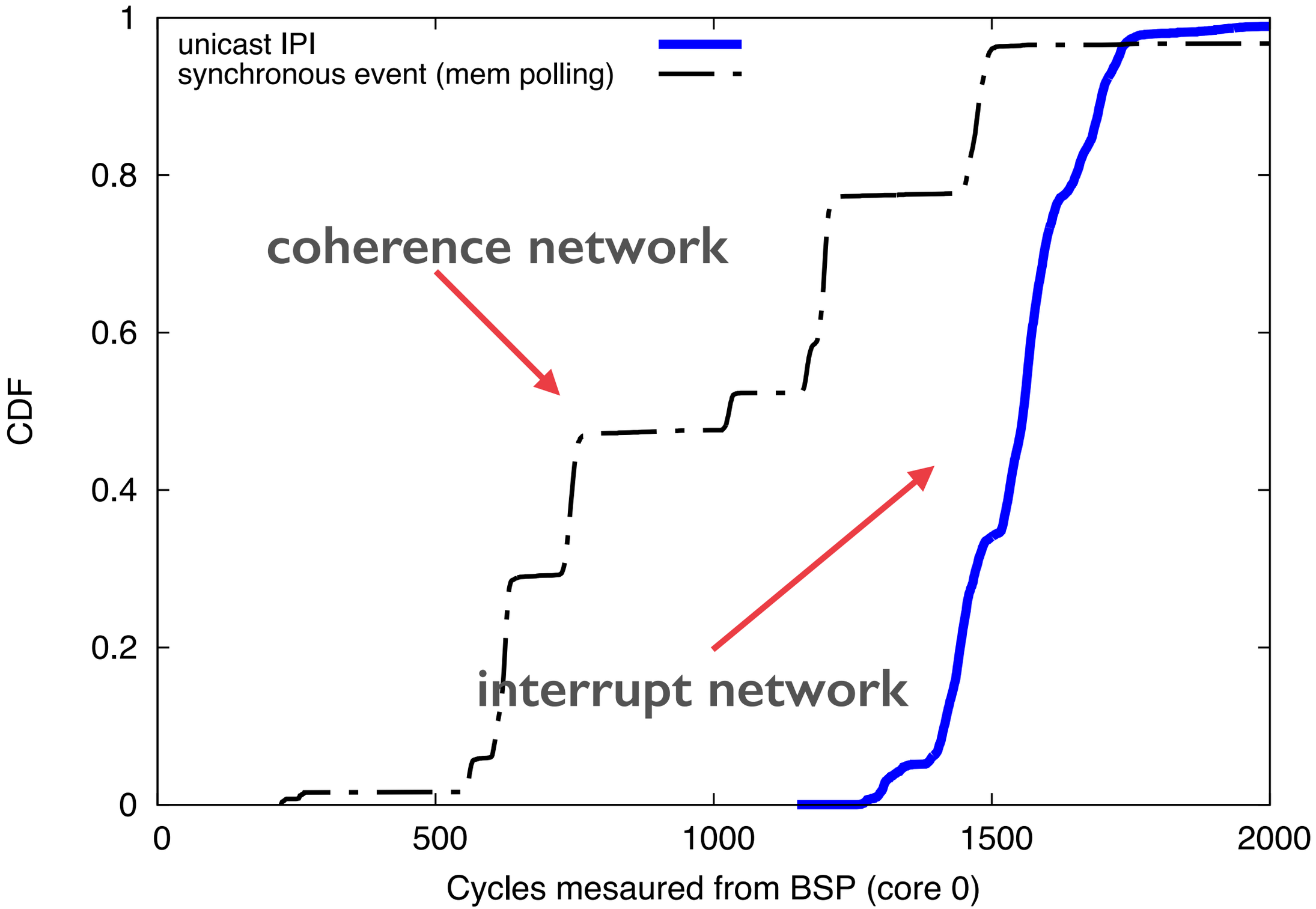
poll!

```
while (!stuff happened)
```



cache invalidation (10s of cycles)

Unicast IPI vs memory polling



WHY THE GAP?

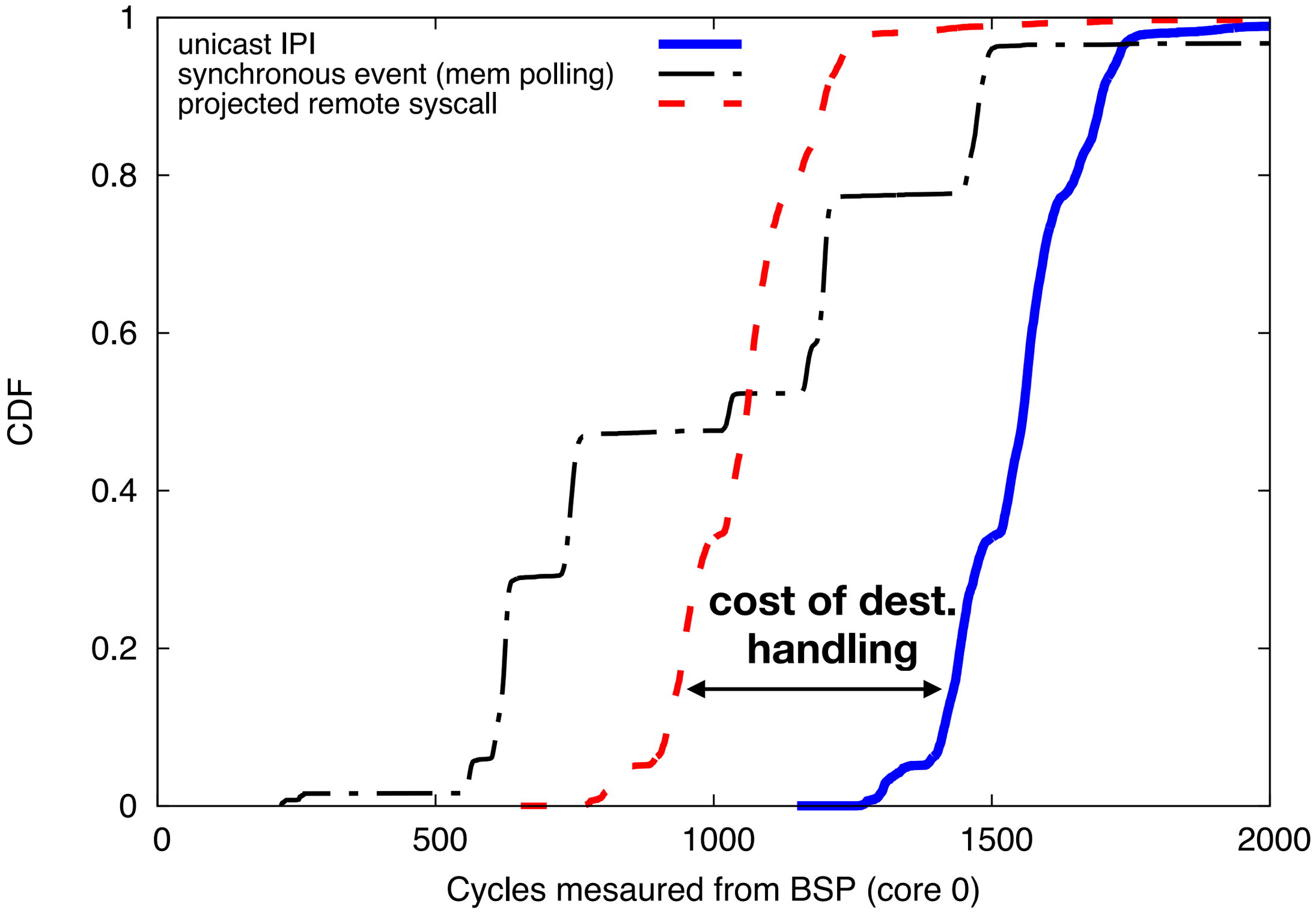
we're bounded by interrupt handling logic in the hardware

IPI cost breakdown

note: these are indirectly measured

we used to have a problem like this with INT80 syscalls…

solution: introduce a new instruction, skip a lot of the interrupt handling logic


unicast IPI
synchronous event (mem polling)
projected remote syscall
cost of dest. handling
CDF
Cycles mesaured from BSP (core 0)
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000