
KubeCon
CloudNativeCon
North America 2024

KubeCon | CloudNativeCon

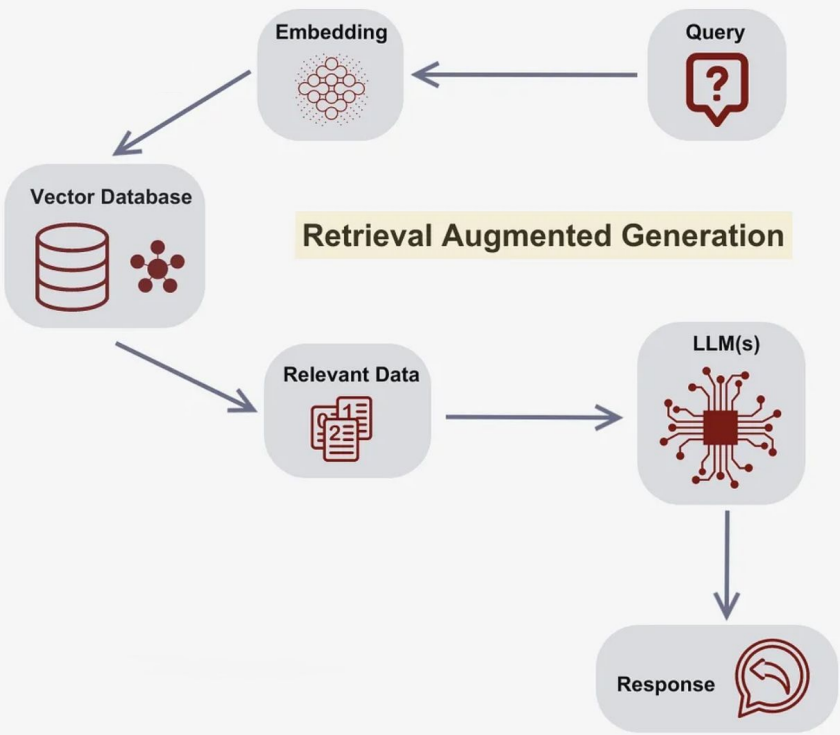
North America 2024

Platform Performance Optimization for AI

a Resource Management Perspective

Dixita Narang @Google
Antti Kervinen @Intel

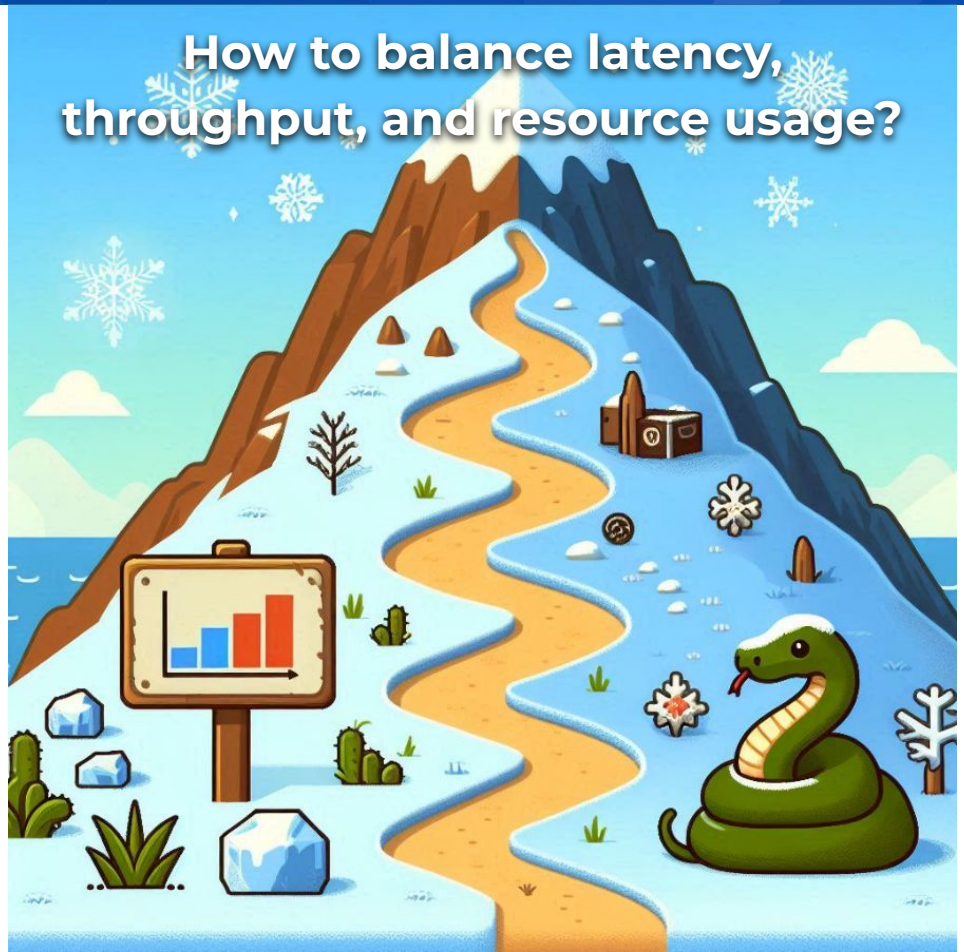
Platform Performance Optimization?



- Example: RAG-pipeline [OPEA ChatQnA]
- Embedding, Database, Re-rank, LLM run as separate microservices in k8s. (Feel free to adjust ReplicaCount!)
- *Platform Performance Optimization*:
  - How to use k8s **node** resources (CPUs, GPUs, memory) optimally?
  - In other words, how to balance latency, throughput, and resource usage?

⇒ Take a component and start measuring!

Goal setting



Story outline

Prepare data collection
Read-only Python instrumentation.

Story outline

Start running tests
Timestamped raw data.

Prepare data collection
Read-only Python instrumentation.

Story outline

Start running tests
Timestamped raw data.

Visualize in the middle of the show
The power of dots.

Prepare data collection
Read-only Python instrumentation.

Story outline

Course correction
Bad idea! New needed!

Start running tests
Timestamped raw data.

Visualize in the middle of the show
The power of dots.

Prepare data collection
Read-only Python instrumentation.

Story outline

Course correction
Bad idea! New needed!

Start running tests
Timestamped raw data.

Reaching the peak
Tables of wisdom.

Visualize in the middle of the show
The power of dots.

Prepare data collection
Read-only Python instrumentation.

Hack Python: on-the-fly instrumentation

main.py

```
import lib
lib.func()
```

lib.py

```
def func():
    print(“called lib.func()”)
```

python3 main.py

```
called lib.func()
```

Hack Python: on-the-fly instrumentation

instrument.py

```
def wrapped_callable(f):
    def wrap(*args, **kwargs):
        print("wrap before call")
        rv = f(*args, **kwargs)
        print("wrap after call")
        return rv
    return wrap

import lib
lib.func = wrapped_callable(lib.func)

sys.argv.pop(0)
runpy.run_module(
    sys.argv[0].replace(".py", ""),
    run_name="__main__")
```

main.py

```
import lib
lib.func()
```

lib.py

```
def func():
    print(“called lib.func()”)
```

python3 instrument.py main.py

```
wrap before func
called lib.func()
wrap after func
```

Hack Python: on-the-fly instrumentation

instrument.py

```
def wrapped_callable(f):
    def wrap(*args, **kwargs):
        print("wrap before call")
        rv = f(*args, **kwargs)
        print("wrap after call")
        return rv
    return wrap

import lib
lib.func = wrapped_callable(lib.func)

sys.argv.pop(0)
runpy.run_module(
    sys.argv[0].replace(".py", ""),
    run_name="__main__")
```

main.py

```
import lib
lib.func()
```

Take away

"Decorate" libraries without touching their code.

You can wrap objects without knowing their classes… (next)

python3 instrument.py main.py

```
wrap before func
called lib.func()
wrap after func
```

Hack Python: on-the-fly instrumentation

mychat.py: (adding a taste of reality)

```
from transformers import AutoTokenizer, AutoModelForCausalLM, TextStreamer

model_name = "EleutherAI/gpt-neo-2.7B"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

streamer = TextStreamer(tokenizer, skip_prompt=True, skip_special_tokens=False)

inputs = tokenizer("What happens at KubeCon?", return_tensors="pt")

model.generate(input_ids=inputs['input_ids'], streamer=streamer, max_length=23)
```

`streamer.put()` will be called many times – how to record timestamps?

Hack Python: on-the-fly instrumentation

instrument.py:

```
…
import transformers
…
transformers.TextStreamer = returned_object_wrapper(
    transformers.TextStreamer,
    "put",
    call_timestamp_wrapper,
    key="streamer.put")
…
```

For a working example, check out transformers under:
https://github.com/askervin/kubecon-na-2024/tree/main/python-instrument

Story outline

Start running tests
Timestamped raw data.

Prepare data collection
Read-only Python instrumentation.

Collect and timestamp system data

- /proc/PID/{numa_maps,status}
- Tip: check out PCM - Performance Counter Monitor
  - local/remote memory bandwidth, cache hits/misses/occupancy, PCI bandwidths

```
OS_ID{socket="0",core="3",thread="1"} 79
Instructions_Retired_Any{socket="0",core="3",thread="1",source="core"} 59695529
Clock_Unhalted_Thread{socket="0",core="3",thread="1",source="core"} 166555735
Clock_Unhalted_Ref{socket="0",core="3",thread="1",source="core"} 124176960
L3_Cache_Misses{socket="0",core="3",thread="1",source="core"} 28017
L3_Cache_Hits{socket="0",core="3",thread="1",source="core"} 335746
L2_Cache_Misses{socket="0",core="3",thread="1",source="core"} 409755
L2_Cache_Hits{socket="0",core="3",thread="1",source="core"} 234891
L3_Cache_Occupancy{socket="0",core="3",thread="1",source="core"} 456
Invariant TSC{socket="0",core="3",thread="1",source="core"} 27309855713257871
```

- Gathering data in **timestamped raw** format.
  Example: store *token timestamps* instead of *token intervals*.
  Enables matching performance fluctuations with other timestamped data from system, for instance.

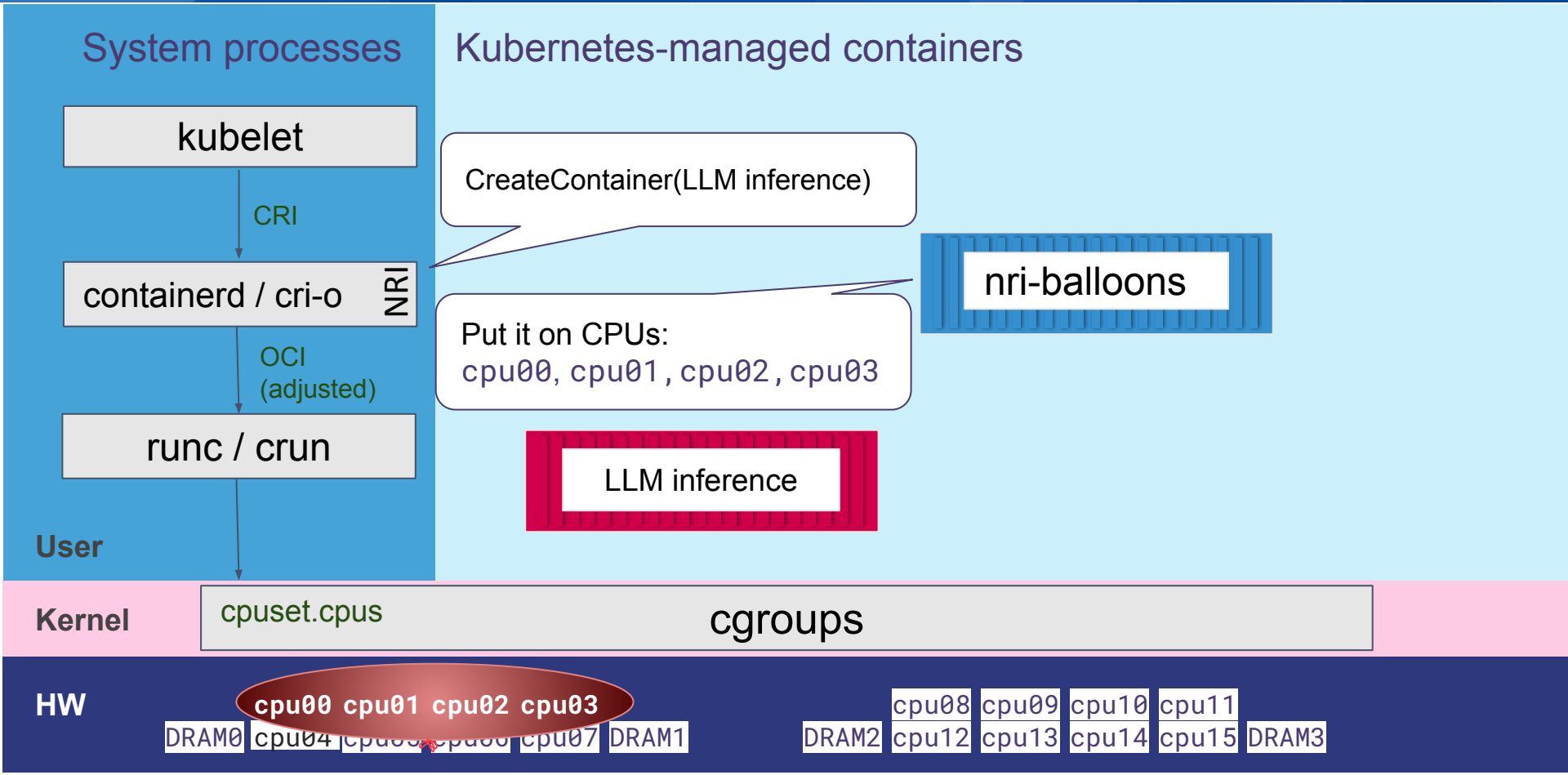
Start running tests

- Run the same LLM prompt in different platform parameter combinations

- Parameter sweep on a 5th Gen Xeon, 2 sockets, 256 CPUs, 1 TB DDR5
  - Number of parallel LLM inference containers: 1, 2, 4, ... , 18 (replicas)
  - Sub-NUMA clustering (BIOS: SNC modes off and SNC2)
  - Number of logical CPUs to allocate for an LLM inference container: 4, 6, ..., 256
  - Allocate hyperthreads from the same or different physical CPU cores?

- How: use the Balloons resource policy from NRI plugins.

Where and how balloons operate



Story outline

Start running tests
Timestamped raw data.

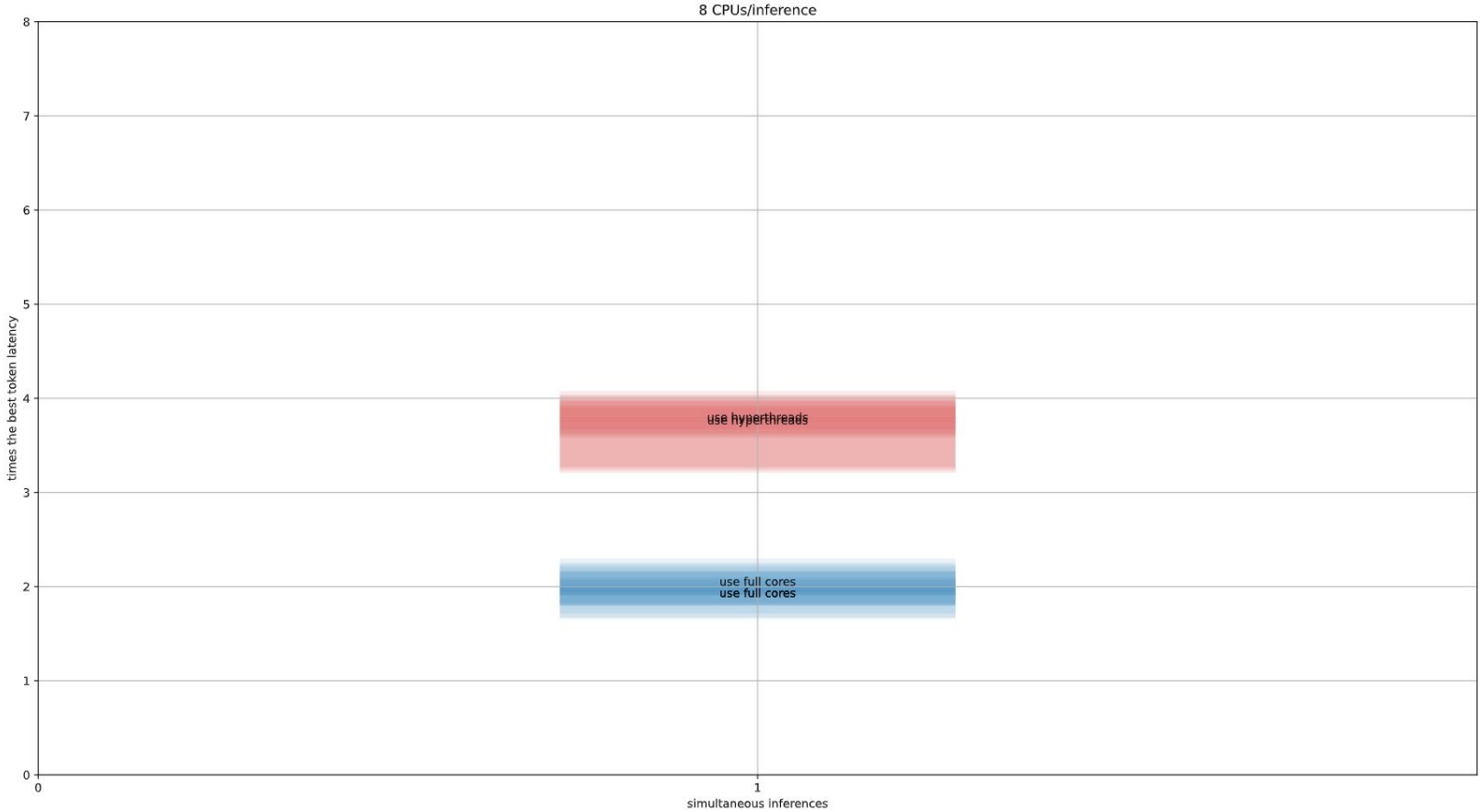
Visualize in the middle of the show
The power of dots.

Prepare data collection
Read-only Python instrumentation.

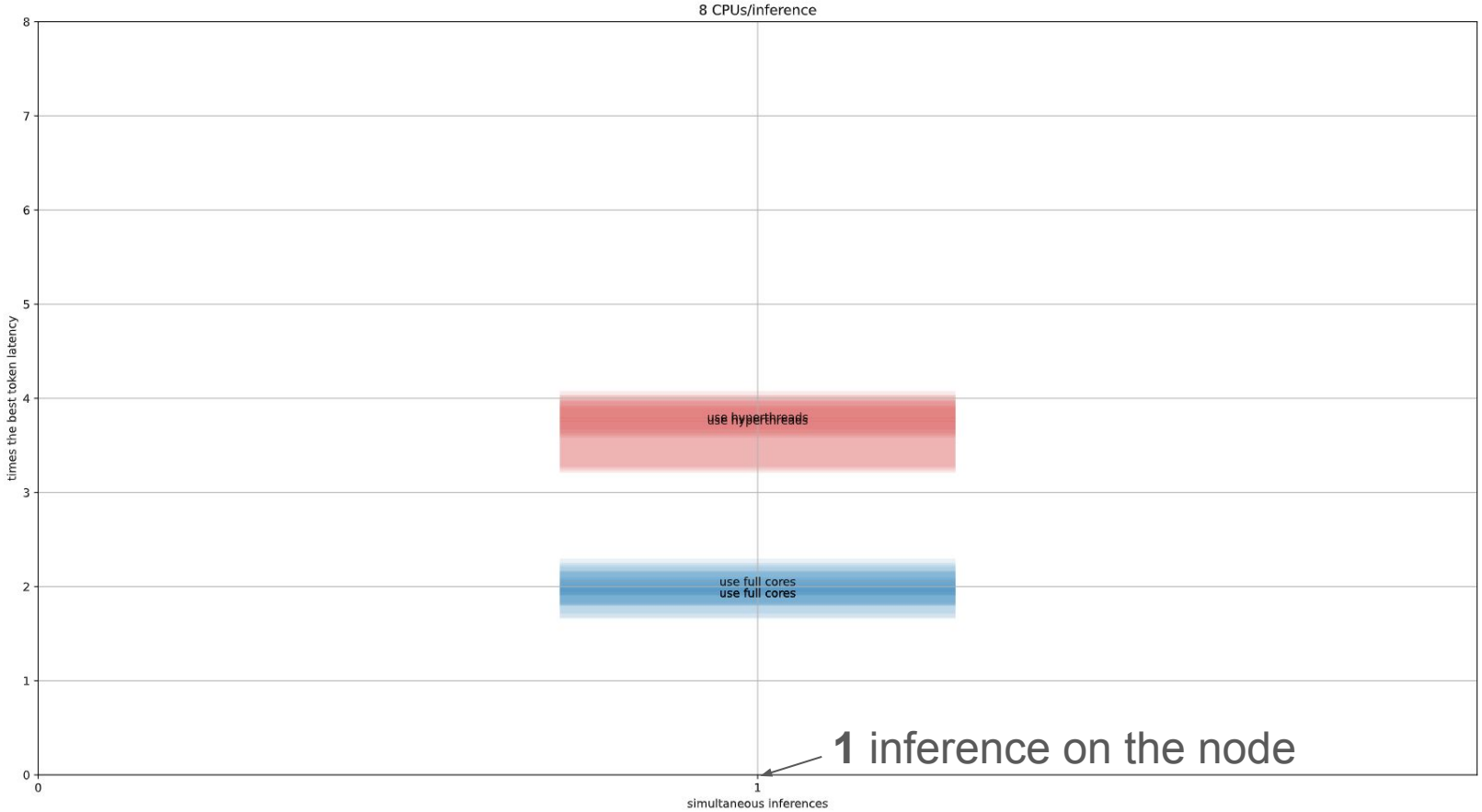
Diving into the data…

For notices, disclaimers, and details about performance claims, visit www.intel.com/PerformanceIndex or scan the QR code:

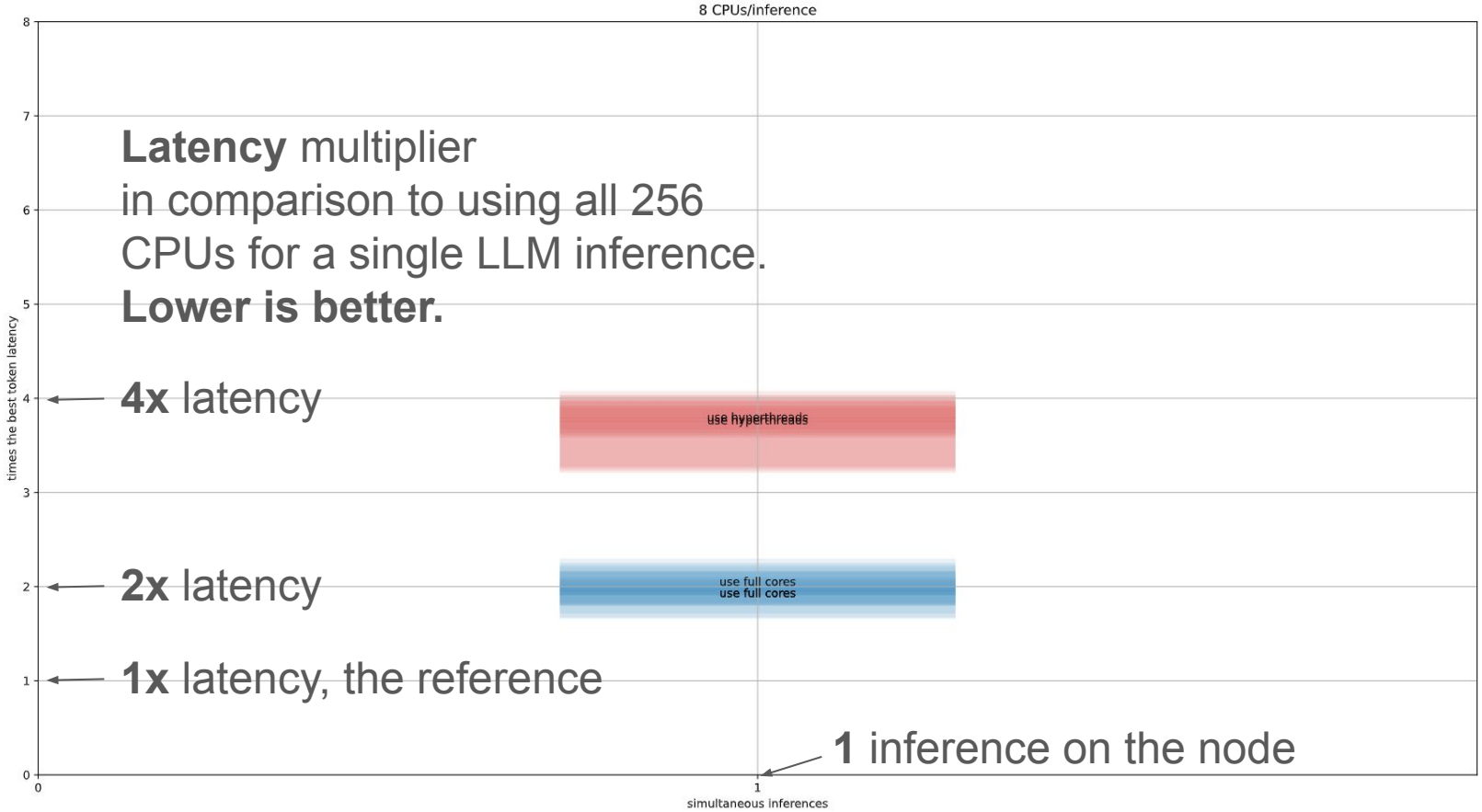
Diving into the data…



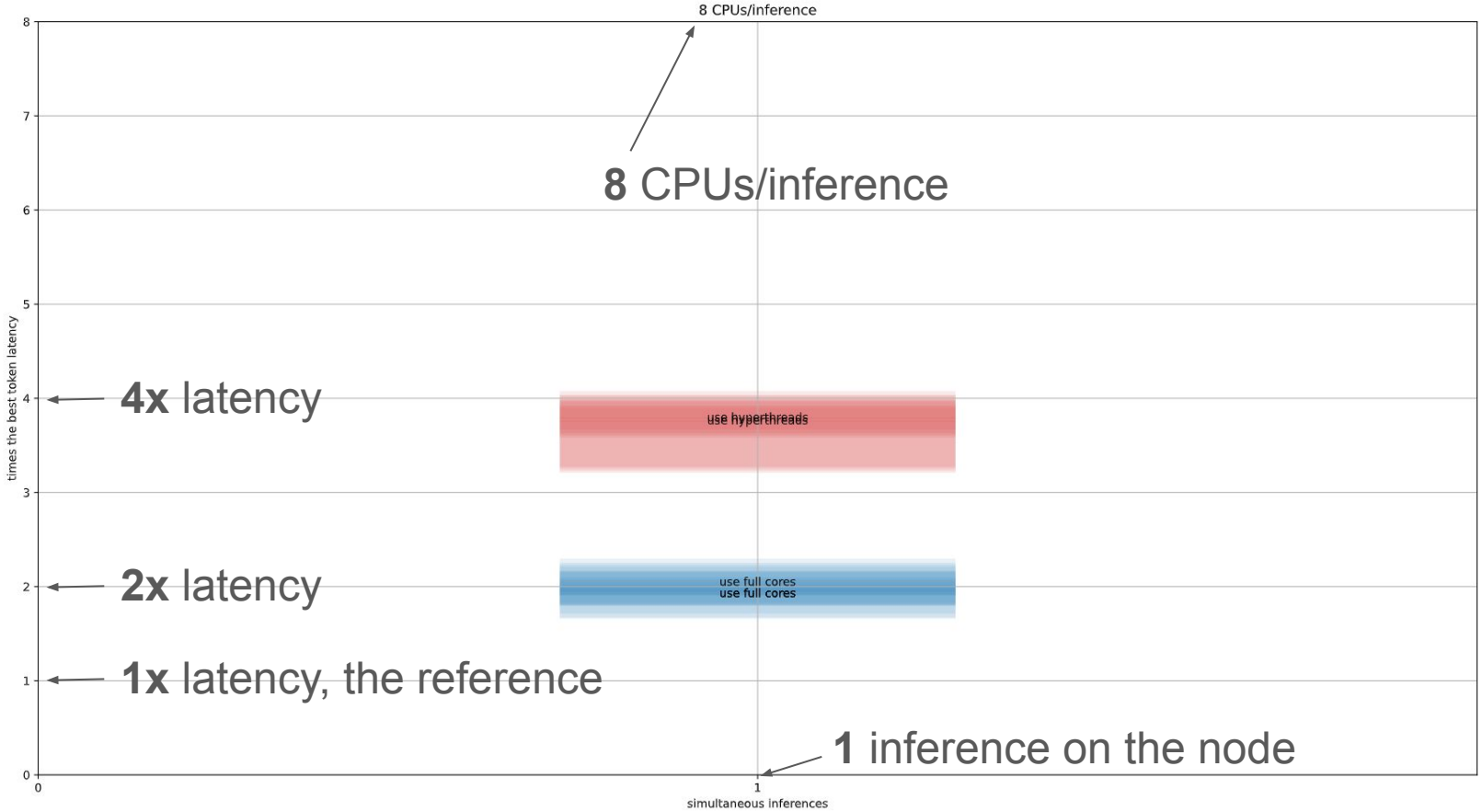
Diving into the data…



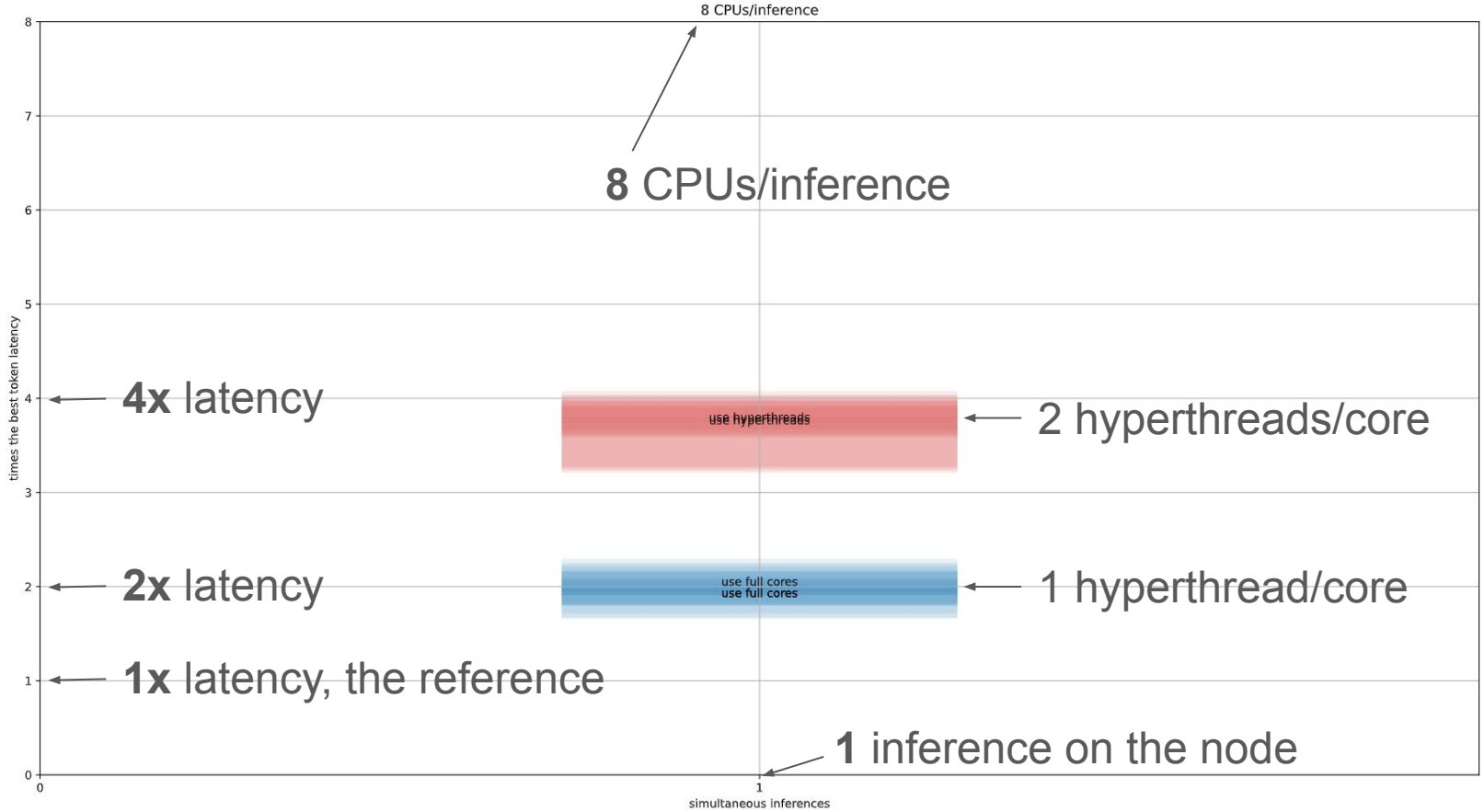
Diving into the data…



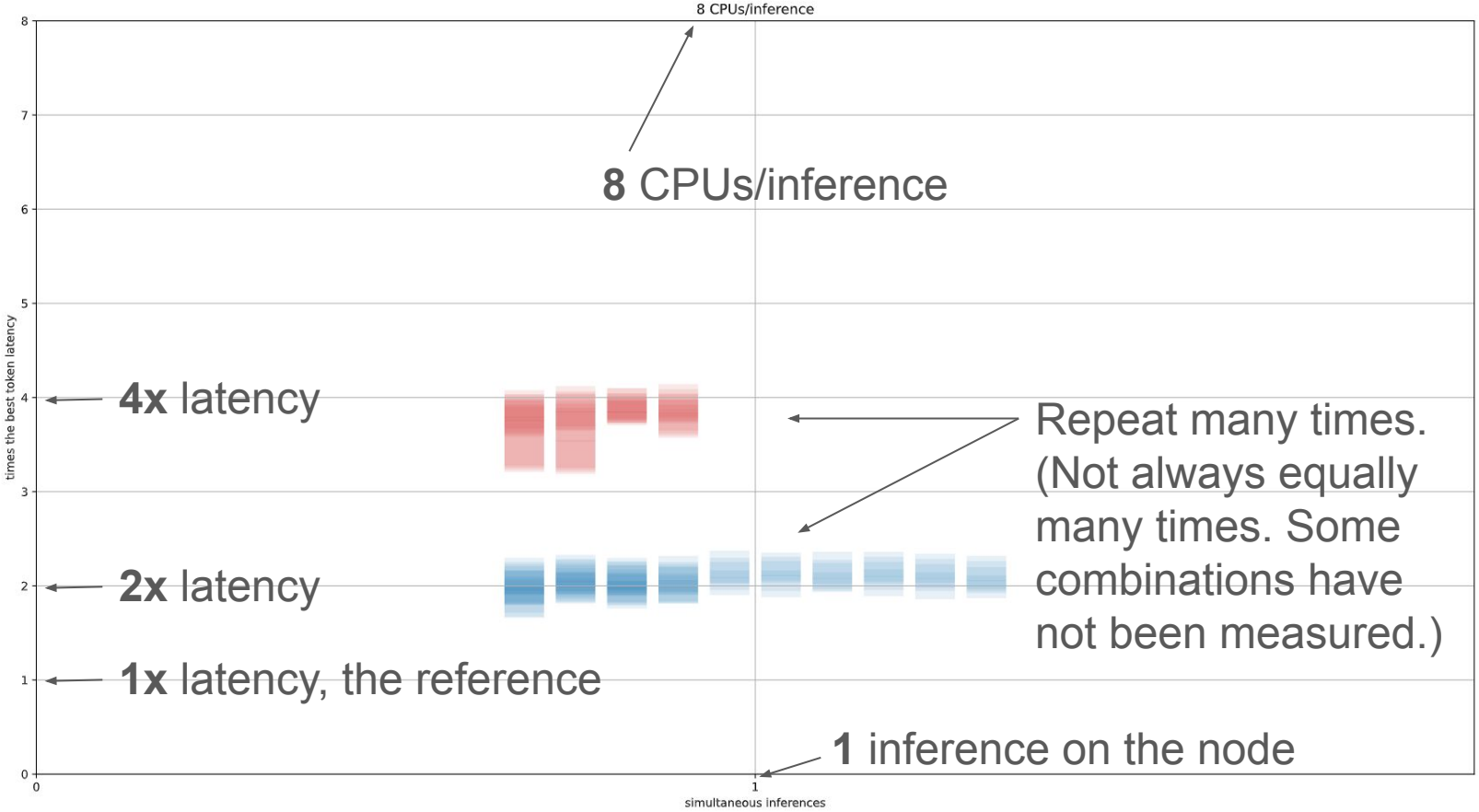
Diving into the data…



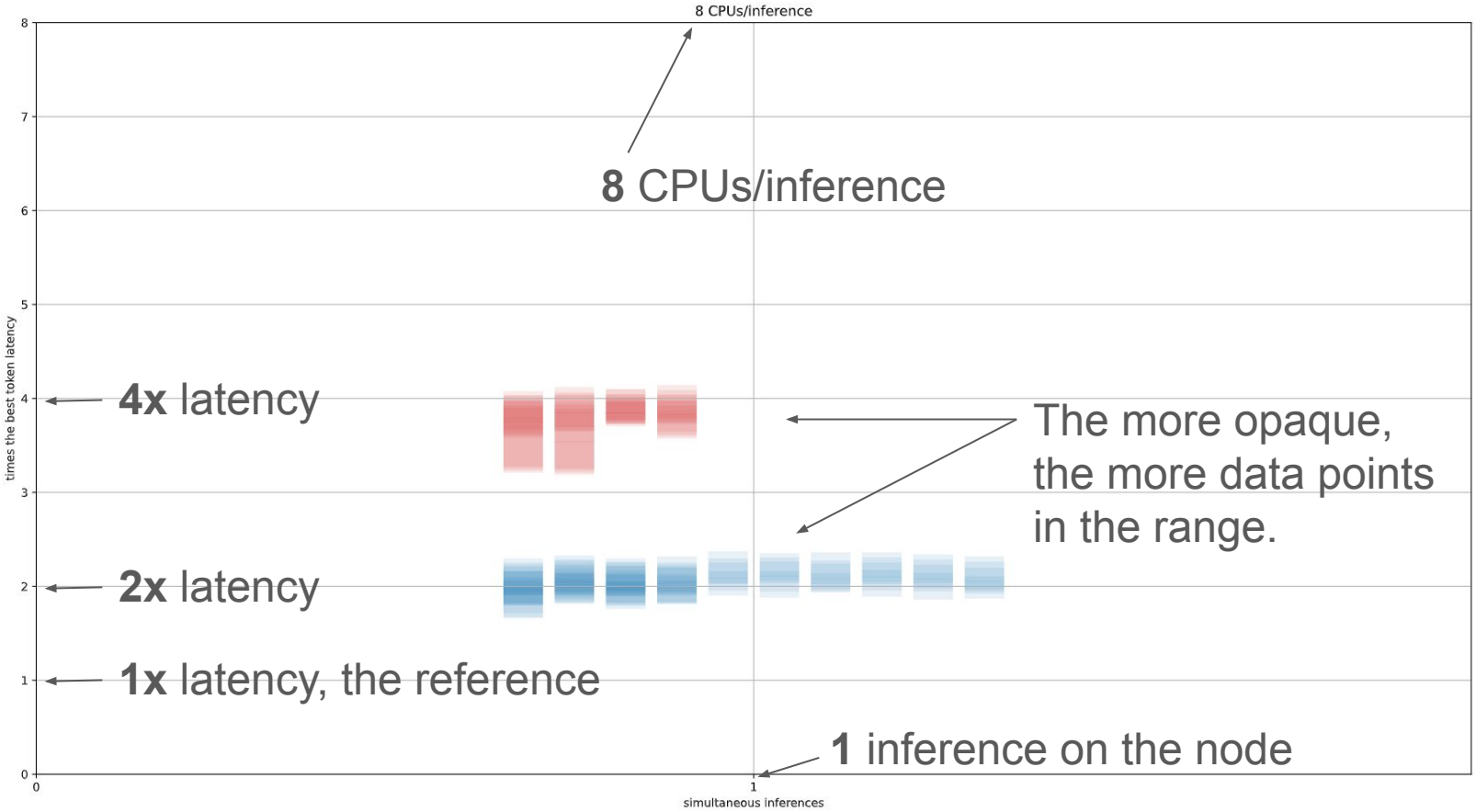
Diving into the data…



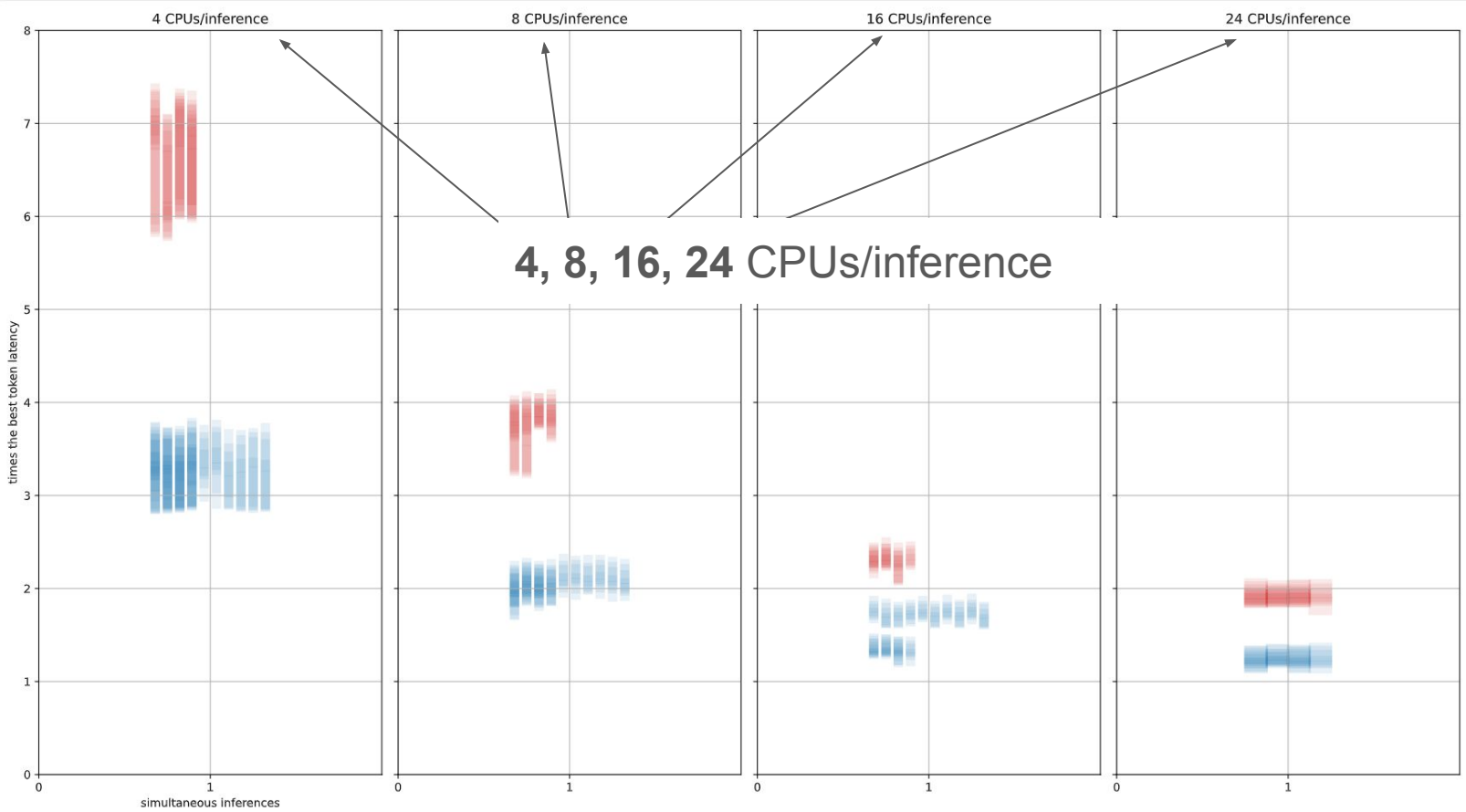
Diving into the data…



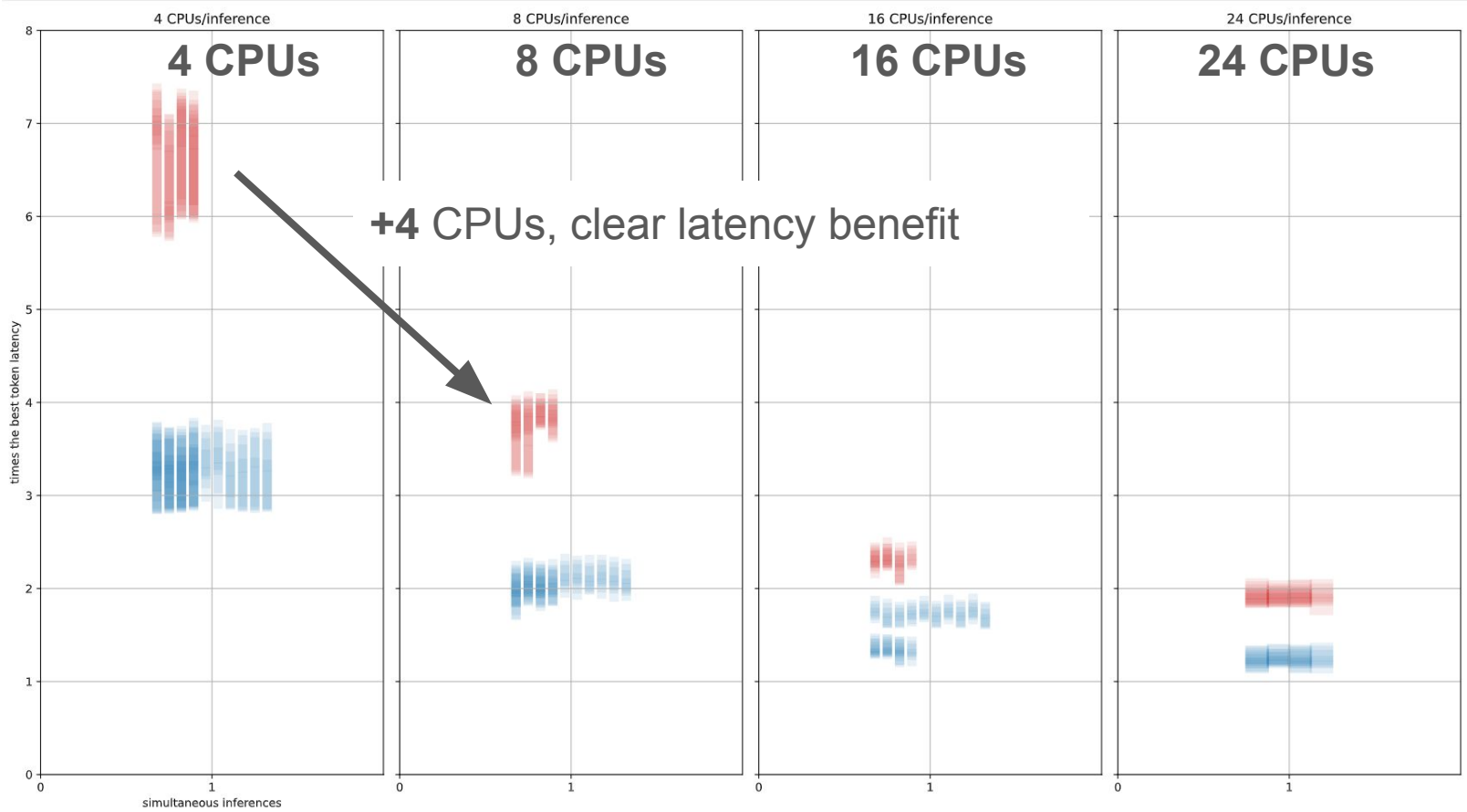
Diving into the data…



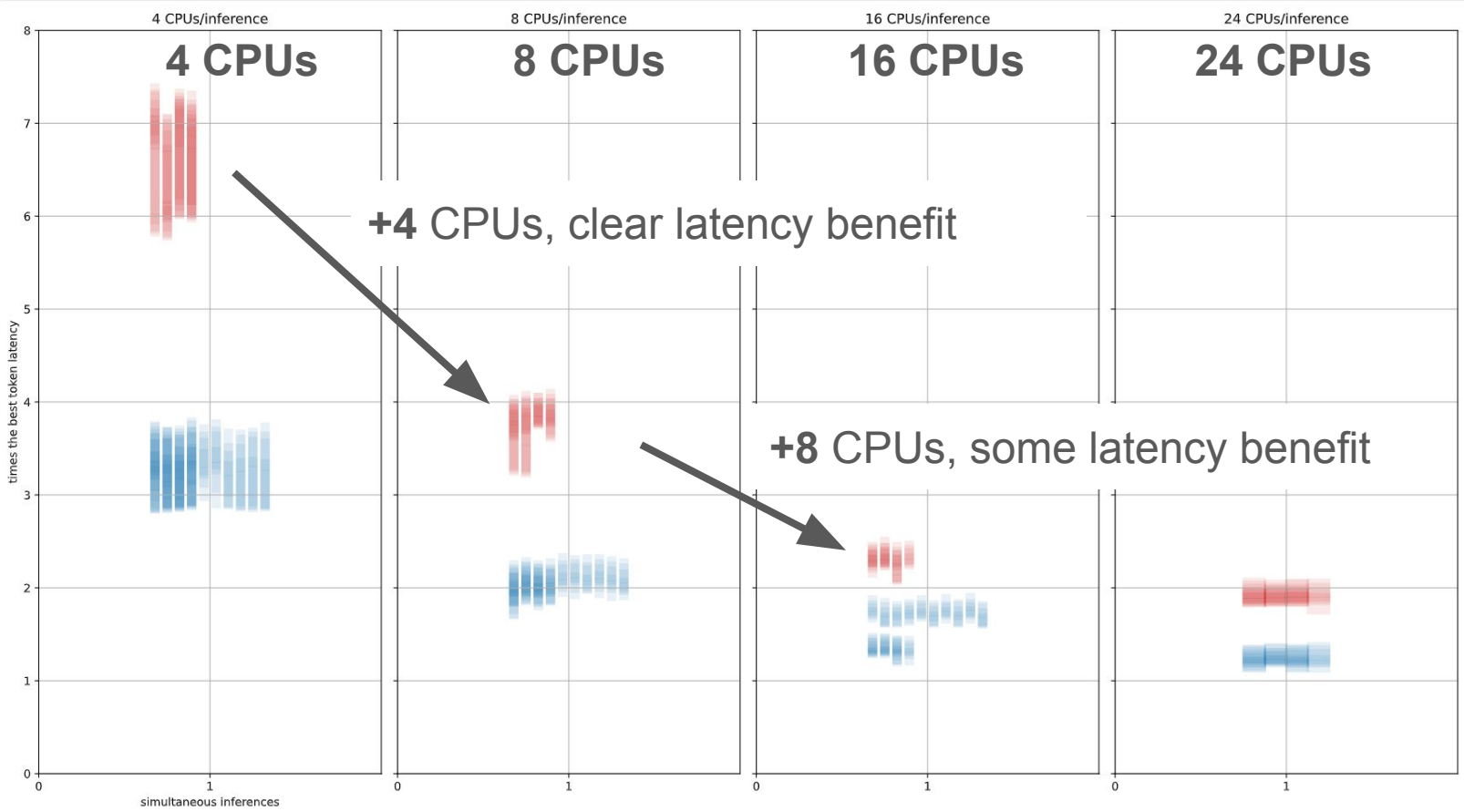
Analyzing #CPUs / inference



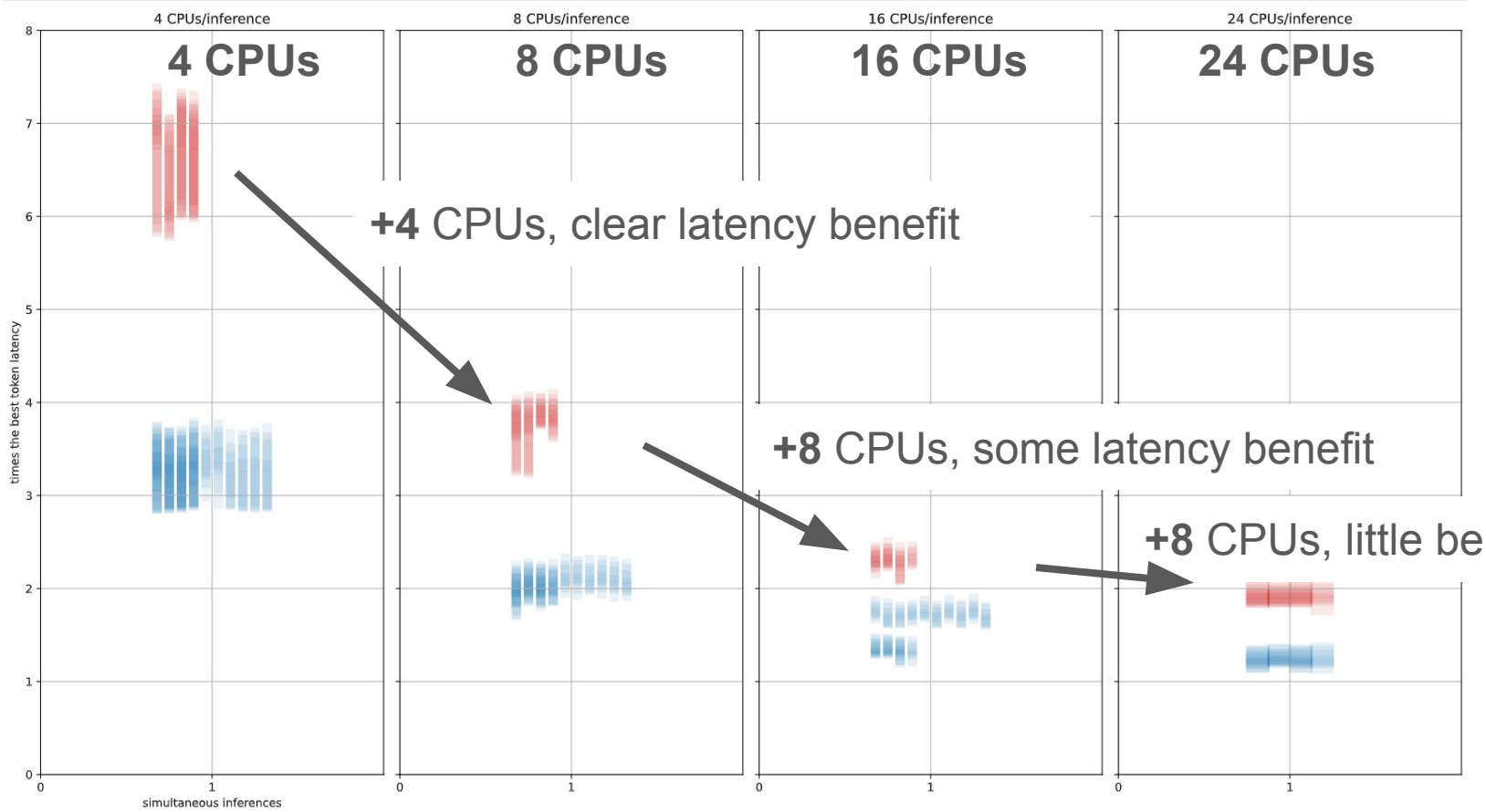
Analyzing #CPUs / inference



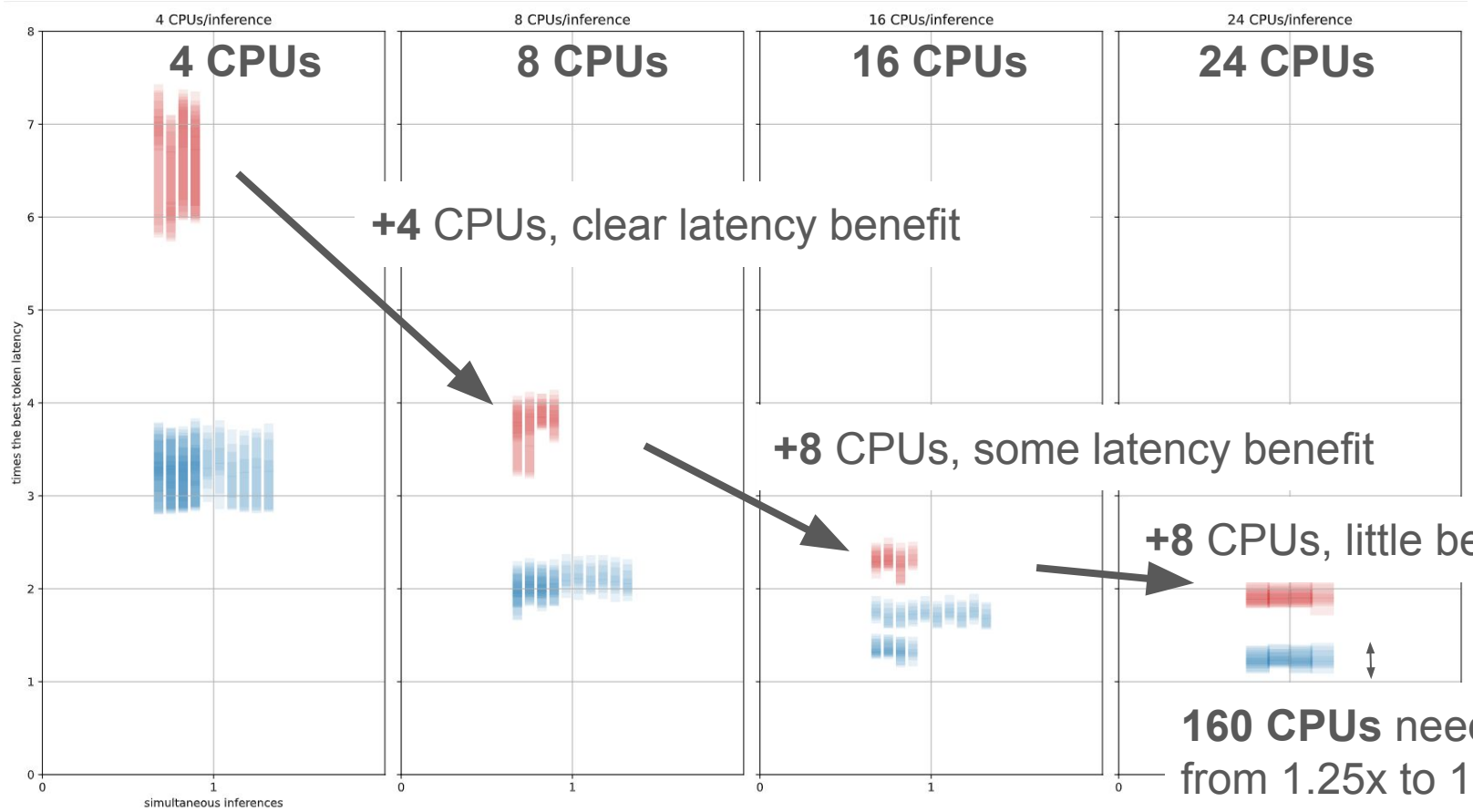
Analyzing #CPUs / inference



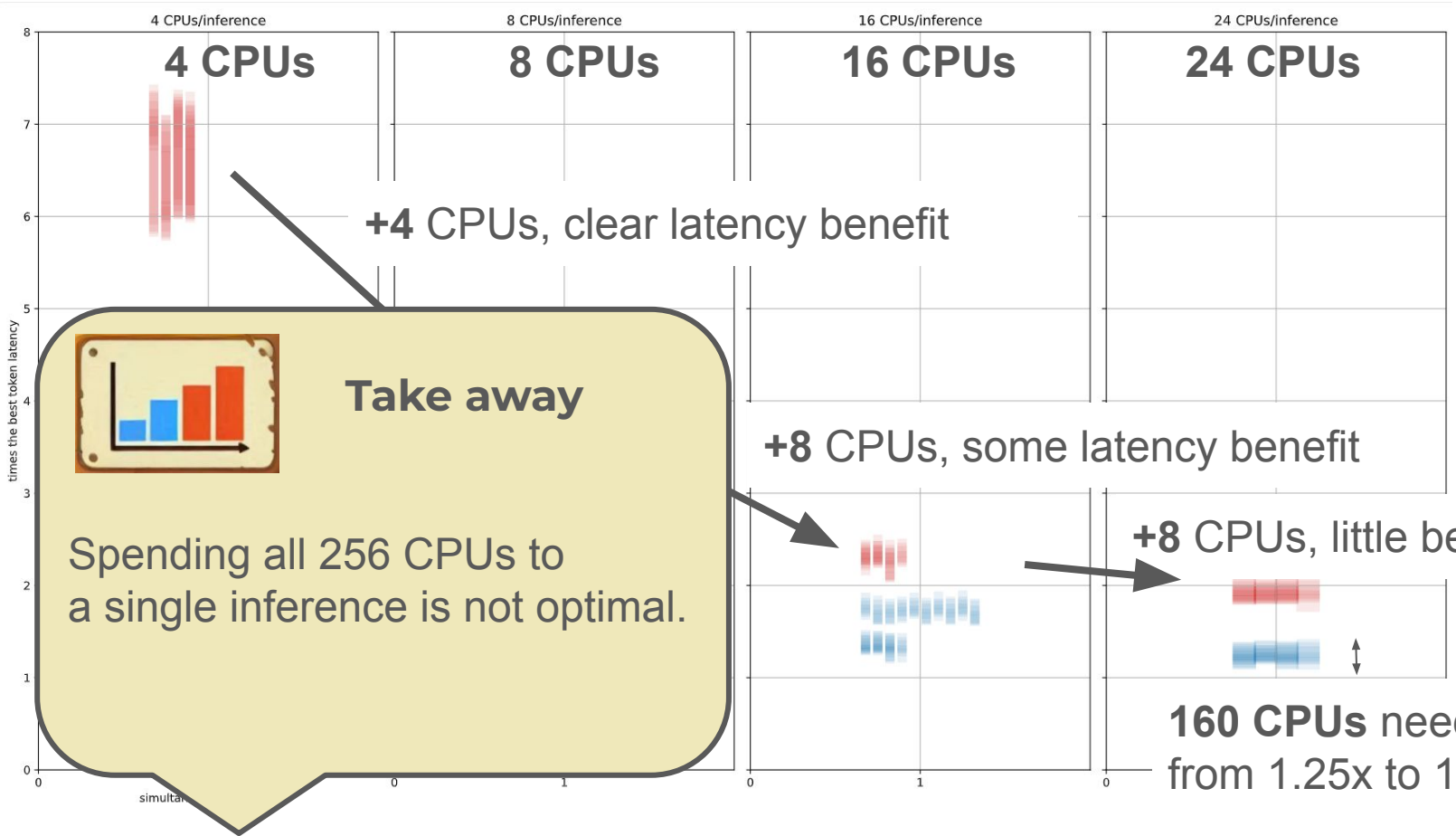
Analyzing #CPUs / inference



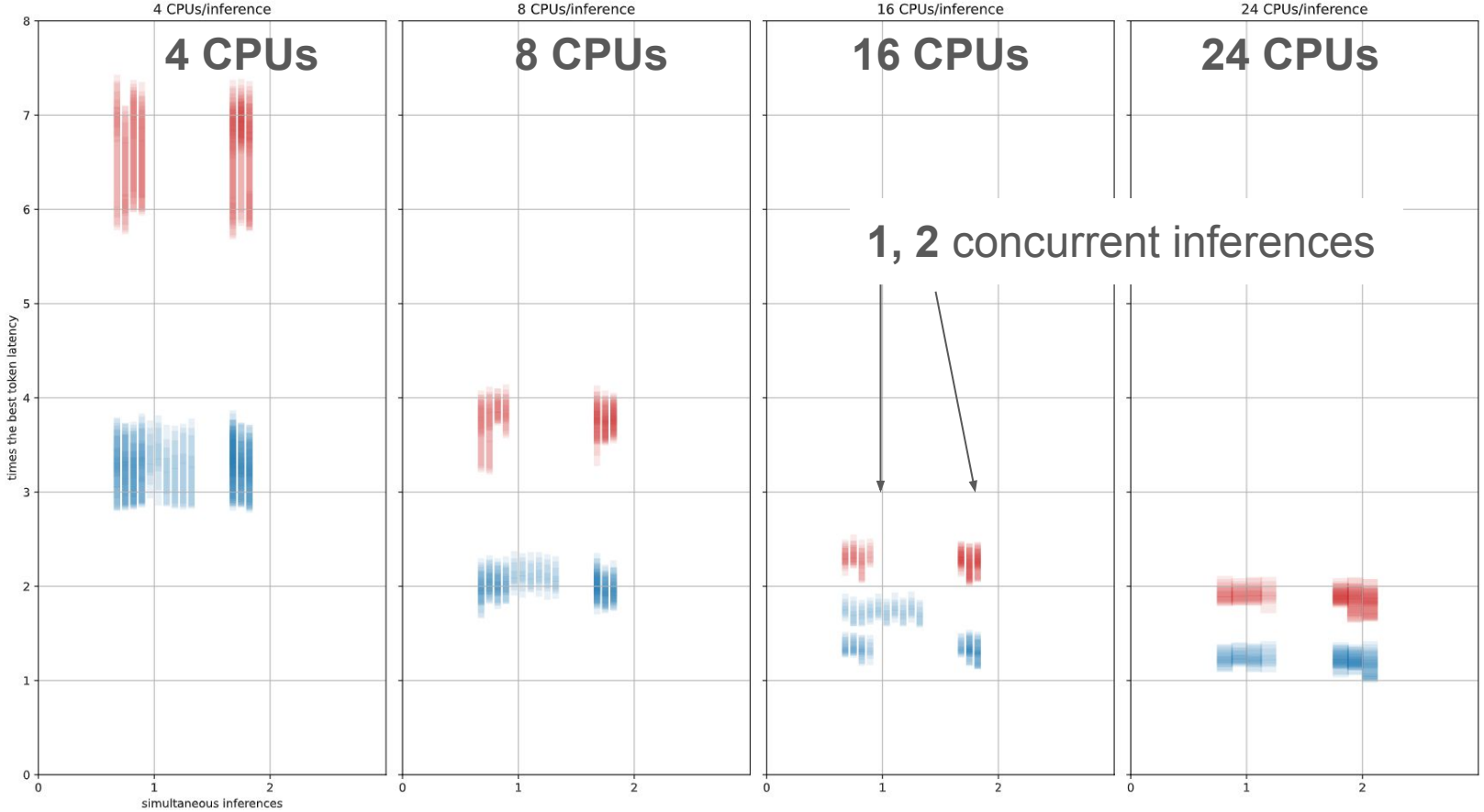
Analyzing #CPUs / inference



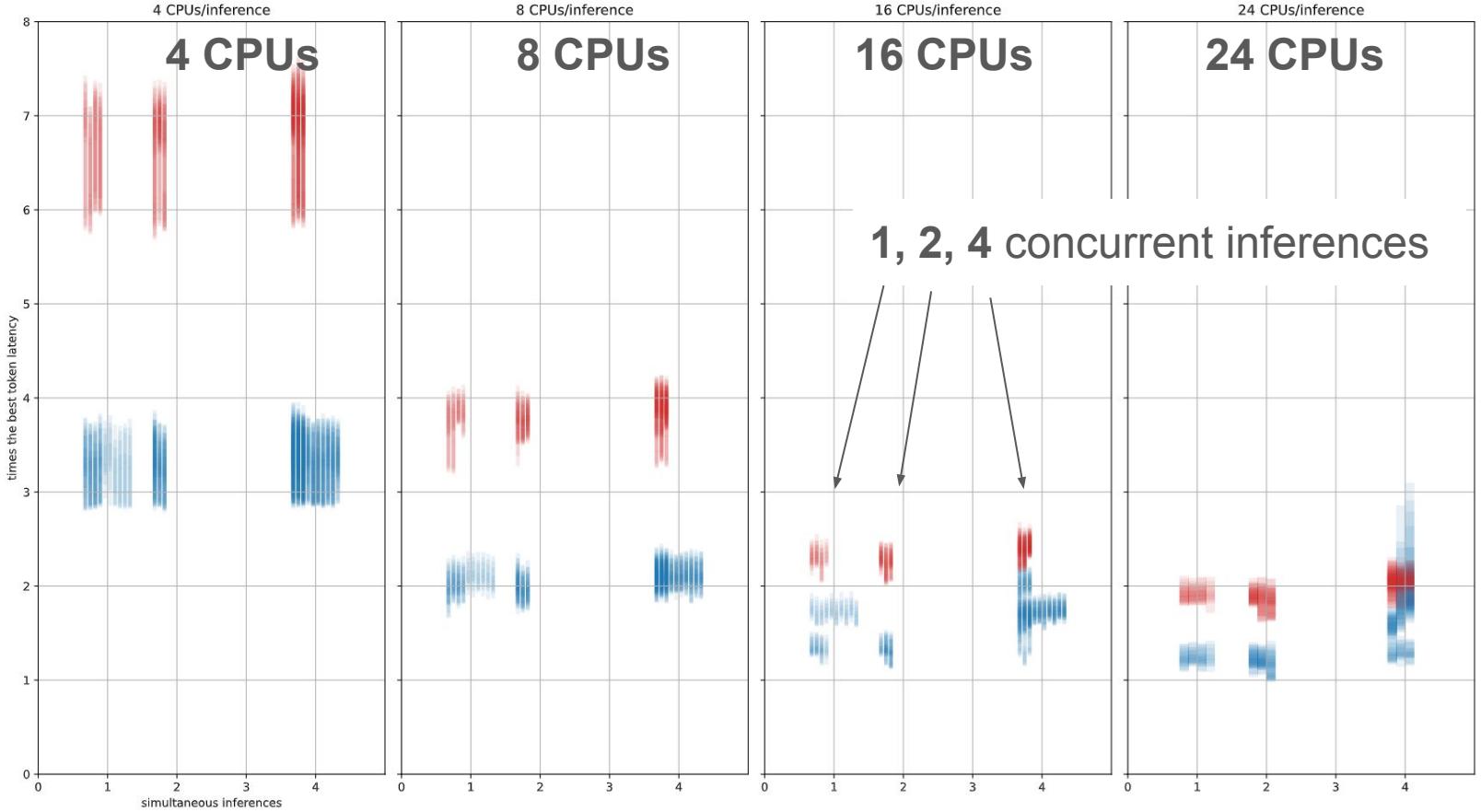
Analyzing #CPUs / inference



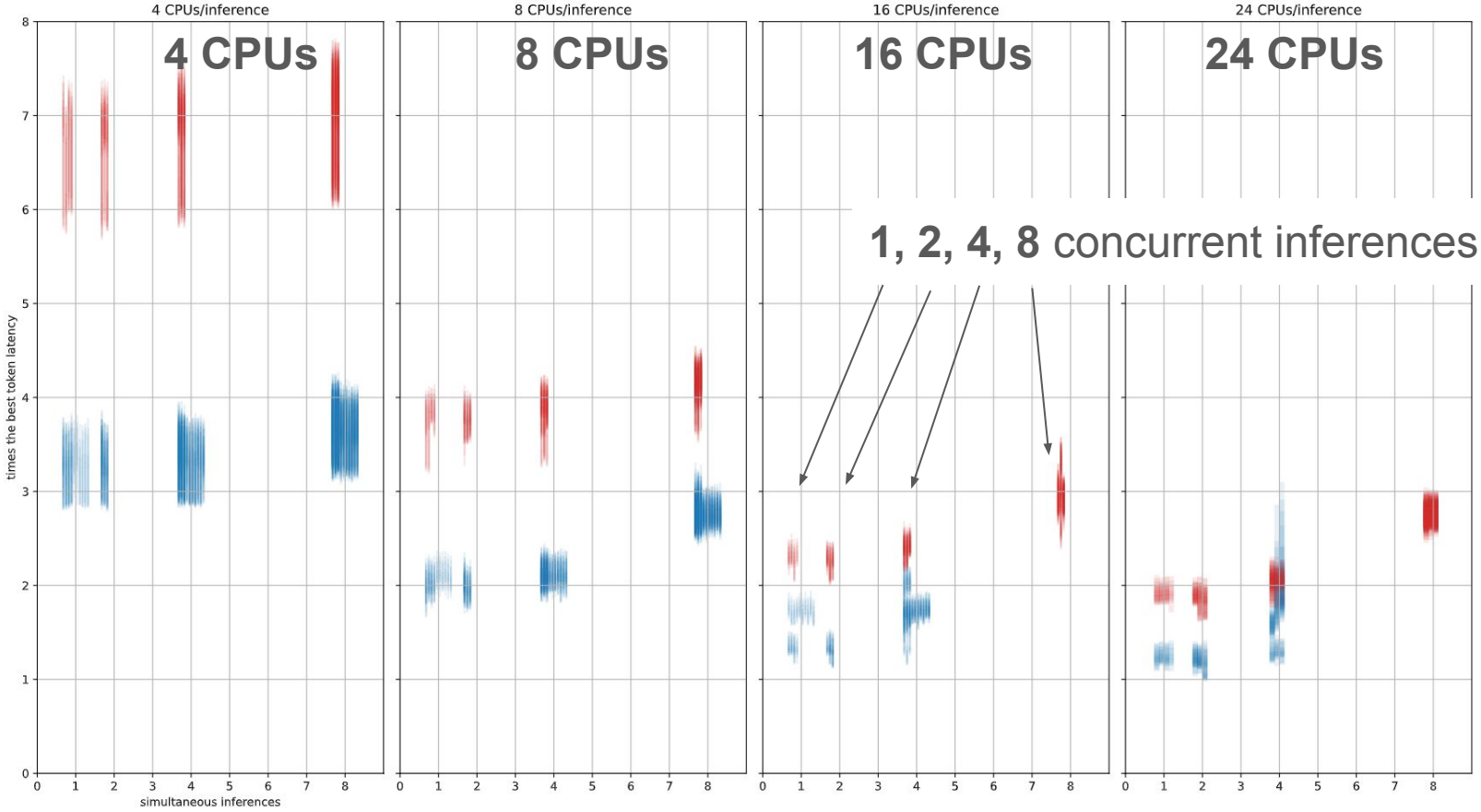
Analyzing #parallel inferences



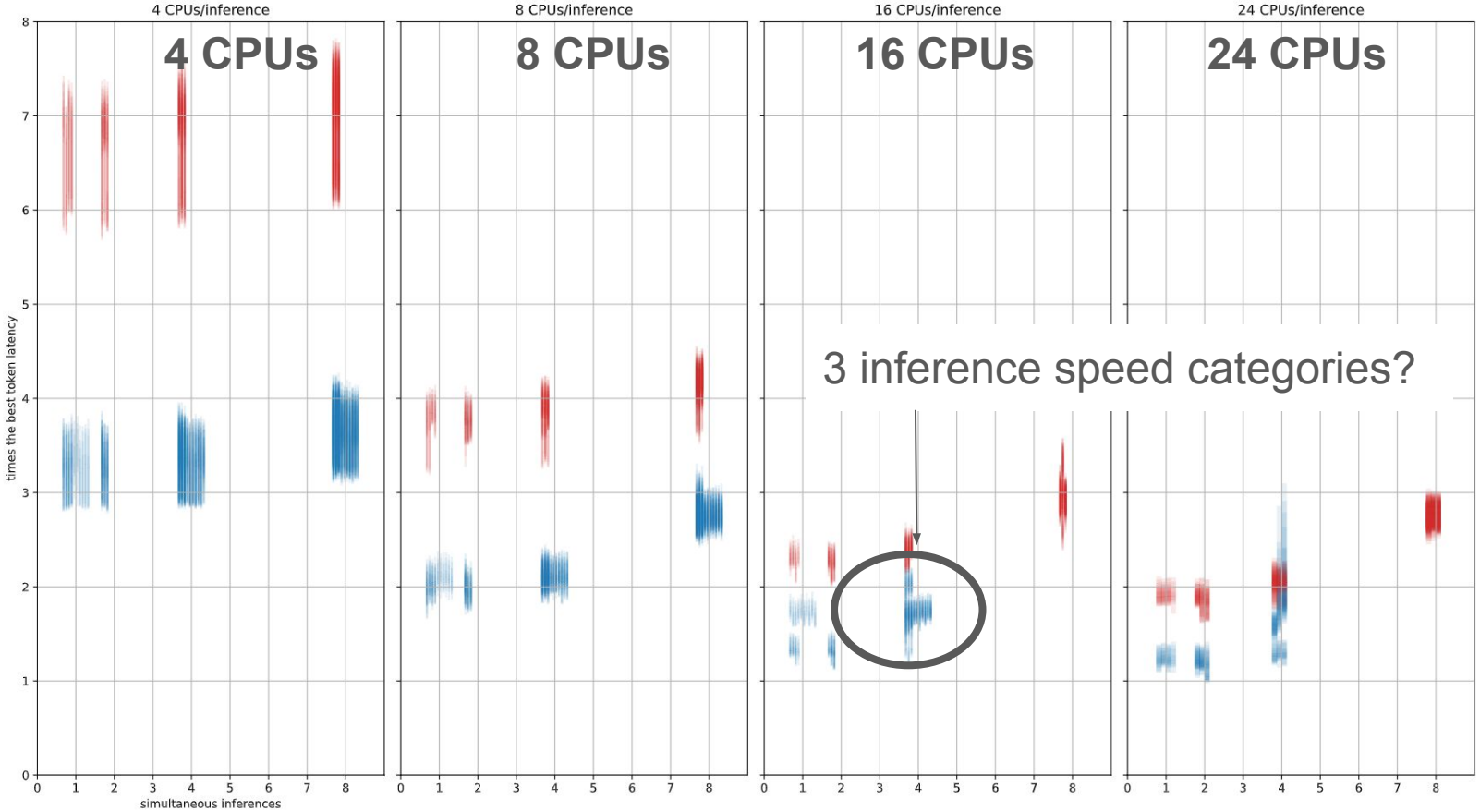
Analyzing #parallel inferences



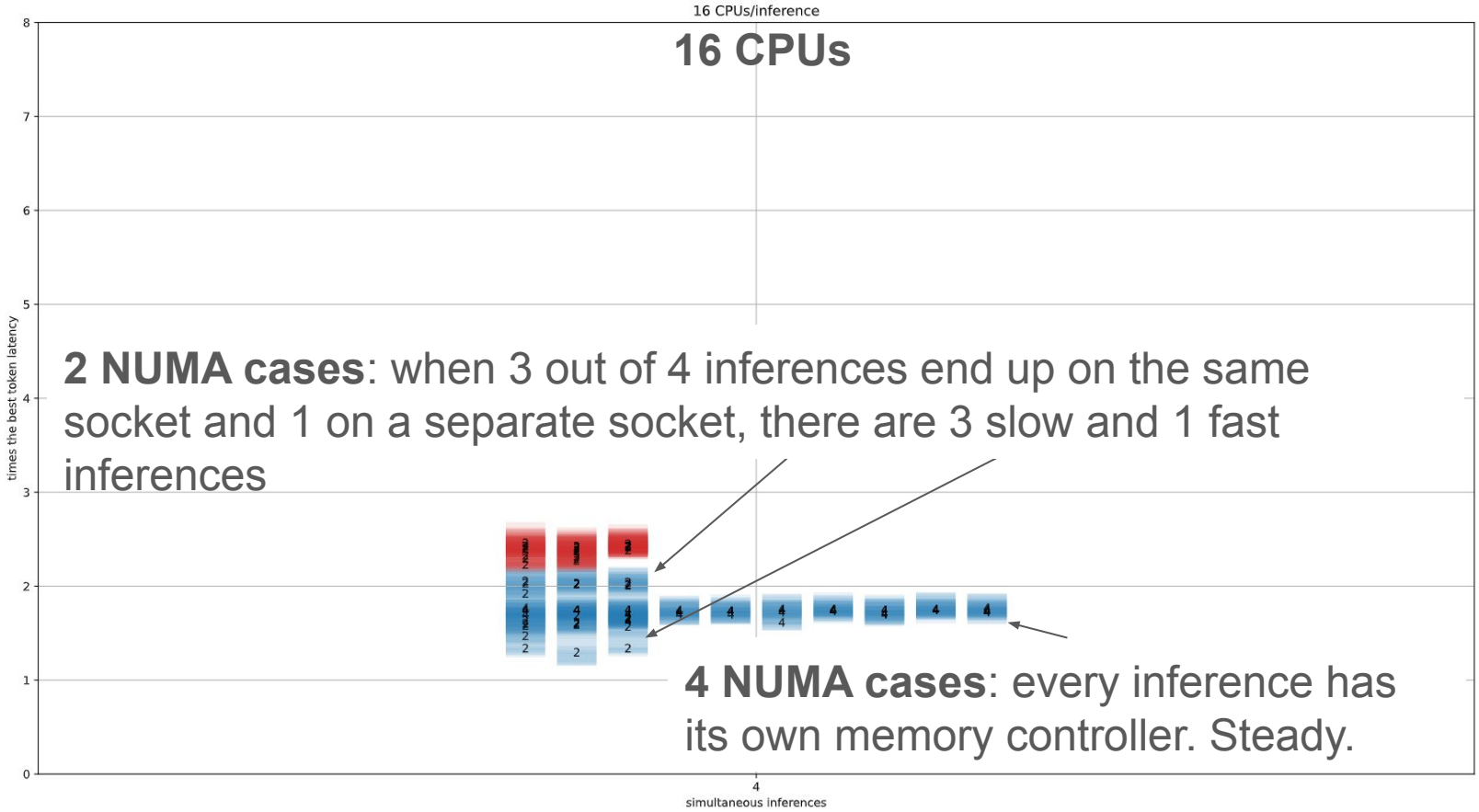
Analyzing #parallel inferences



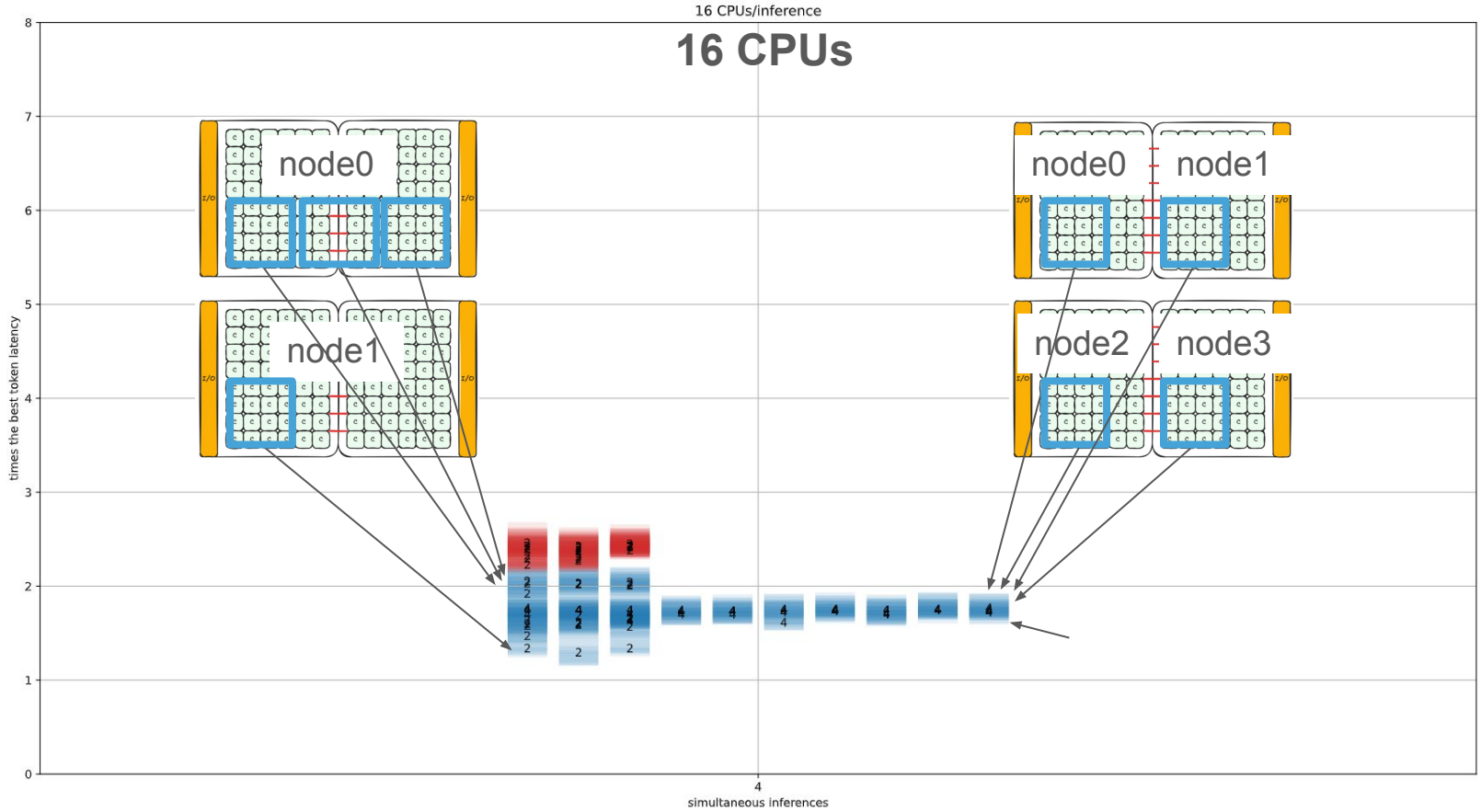
Analyzing #parallel inferences



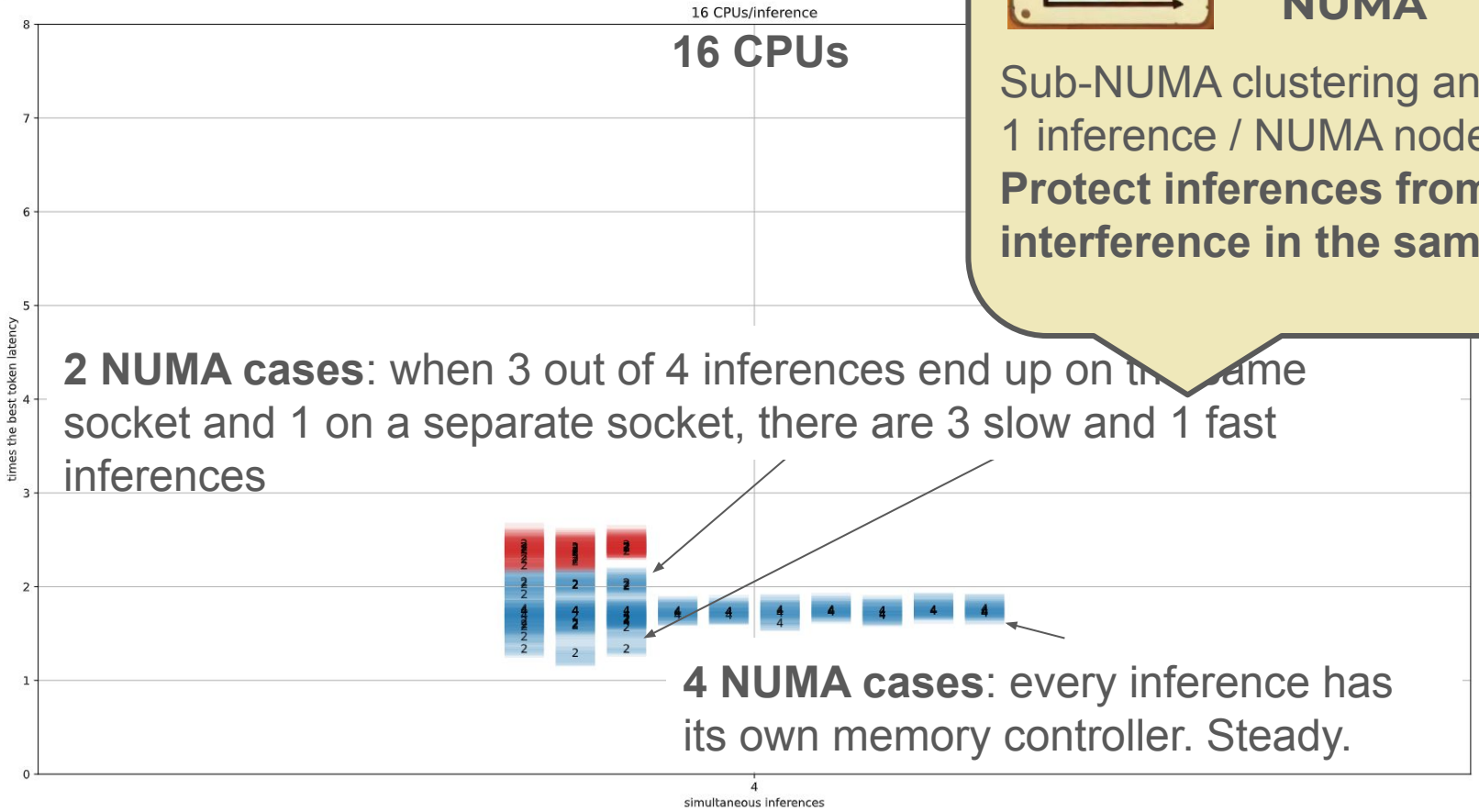
Analyzing #parallel inferences



Analyzing #parallel inferences



Analyzing #parallel inferences



2 NUMA cases: when 3 out of 4 inferences end up on the same socket and 1 on a separate socket, there are 3 slow and 1 fast inferences

4 NUMA cases: every inference has its own memory controller. Steady.

Take away
NUMA

Sub-NUMA clustering and
1 inference / NUMA node:
Protect inferences from interference in the same socket

Story outline

Course correction
Bad idea! New needed!

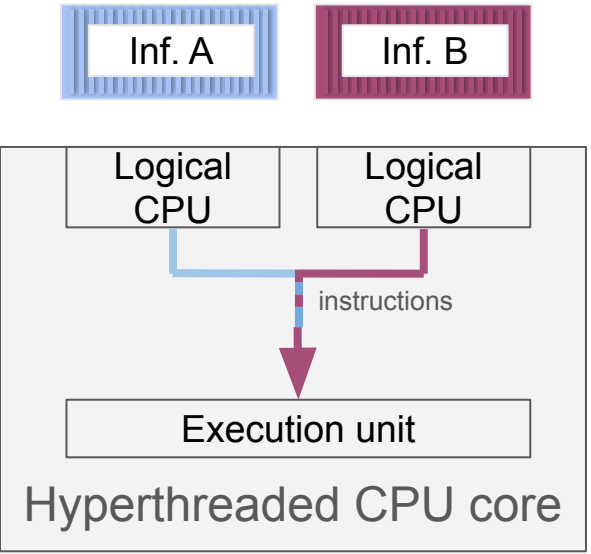
Start running tests
Timestamped raw data.

Visualize in the middle of the show
The power of dots.

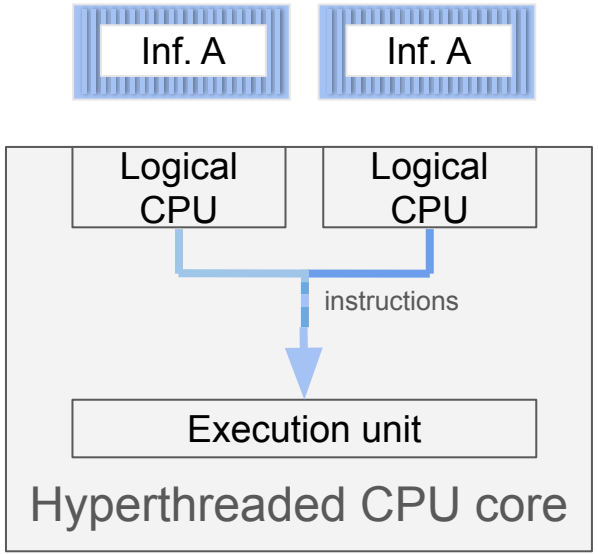
Prepare data collection
Read-only Python instrumentation.

Bad idea for inference when > 50 % CPUs used

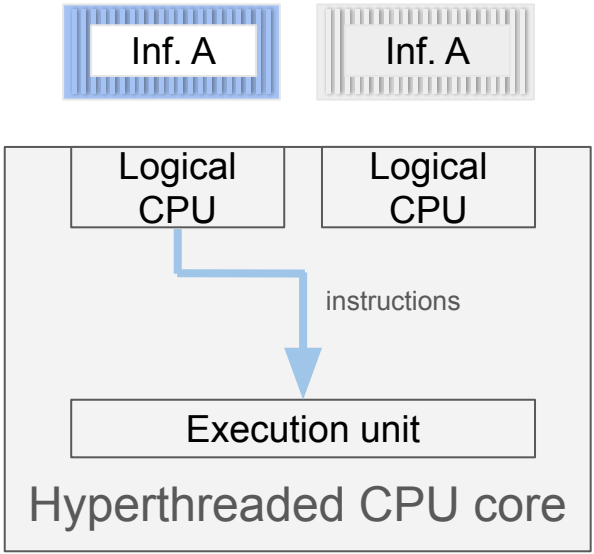
Take logical CPUs from different physical cores.



Take logical CPUs from the same physical cores.

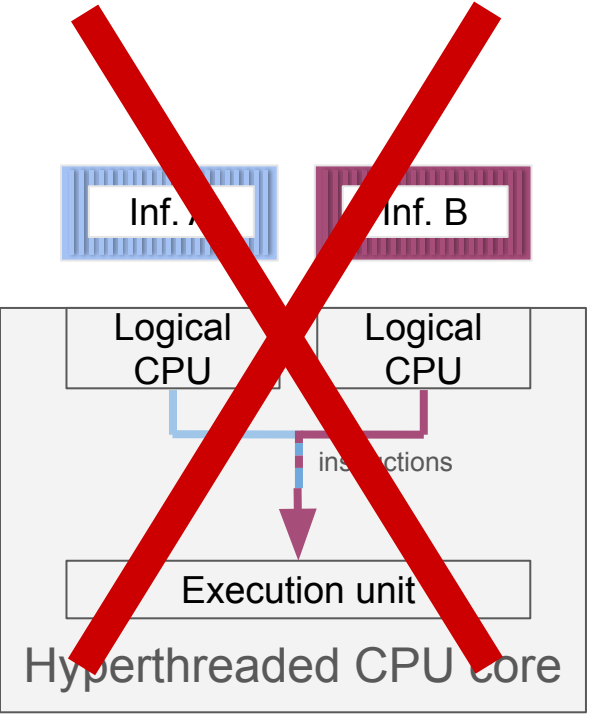


Take logical CPUs from the same physical cores, but use only one.

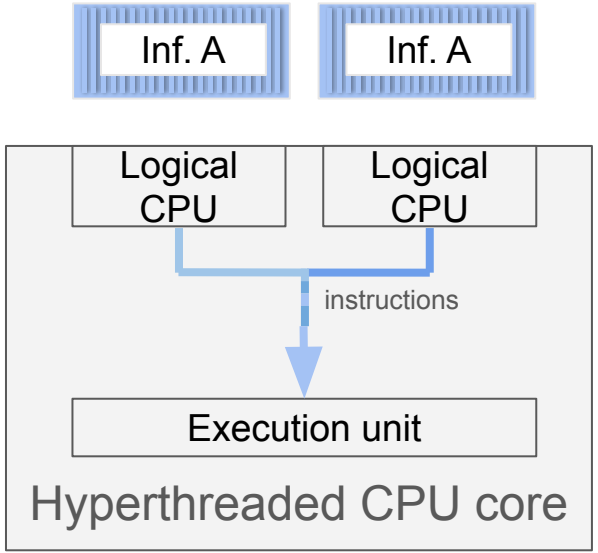


Bad idea for inference when > 50 % CPUs used

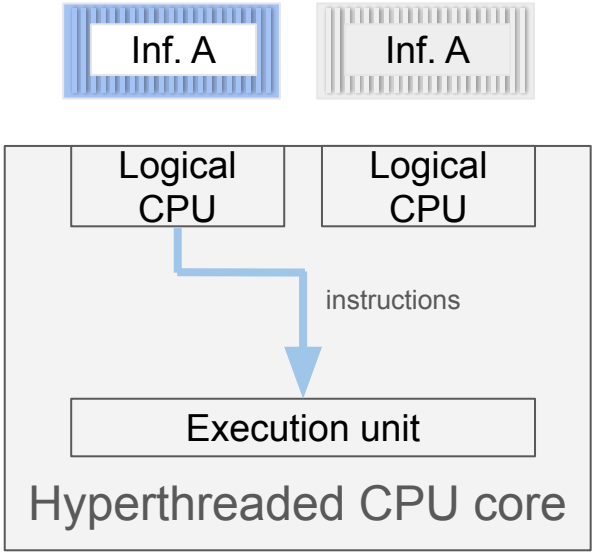
Take logical CPUs from different physical cores.



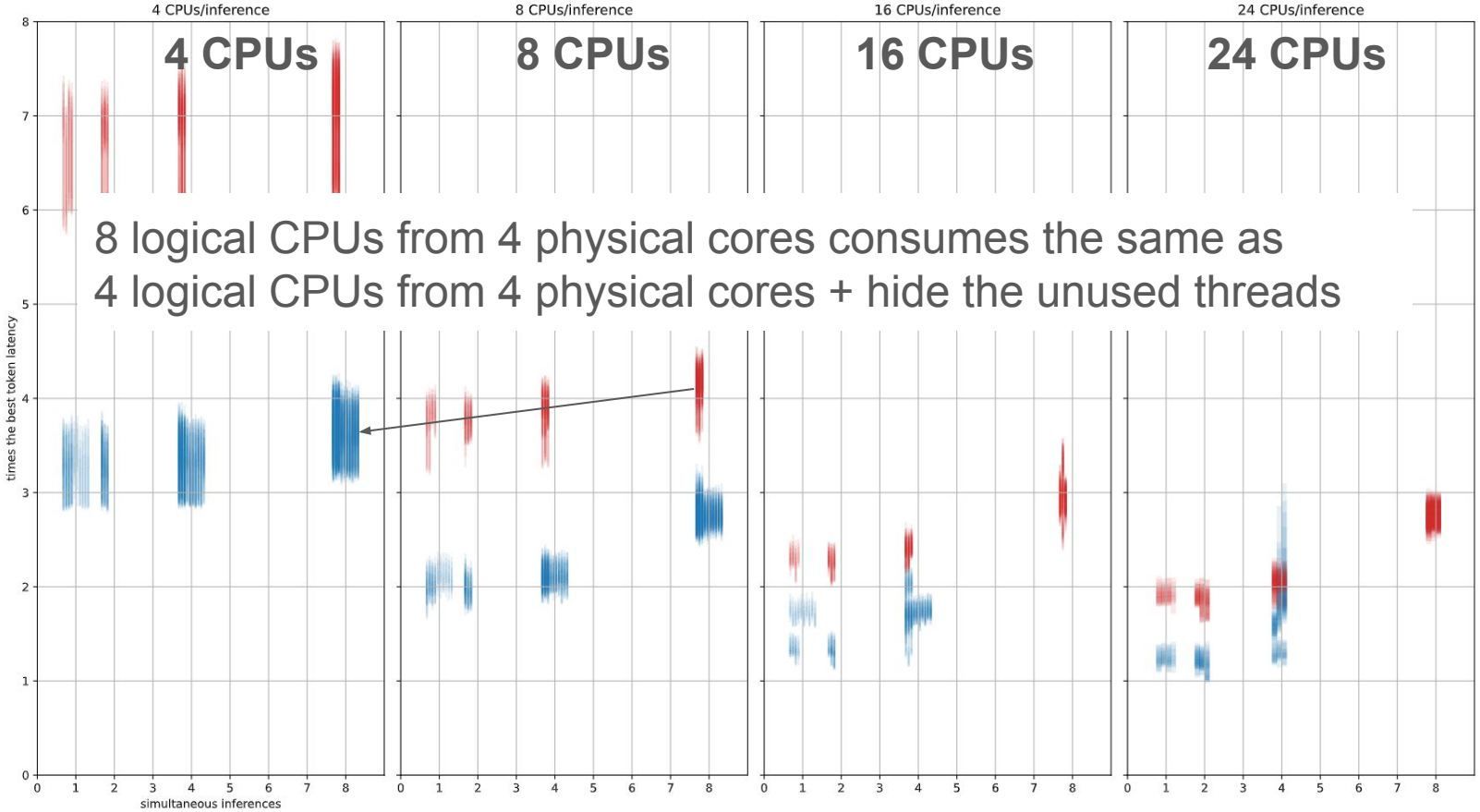
Take logical CPUs from the same physical cores.



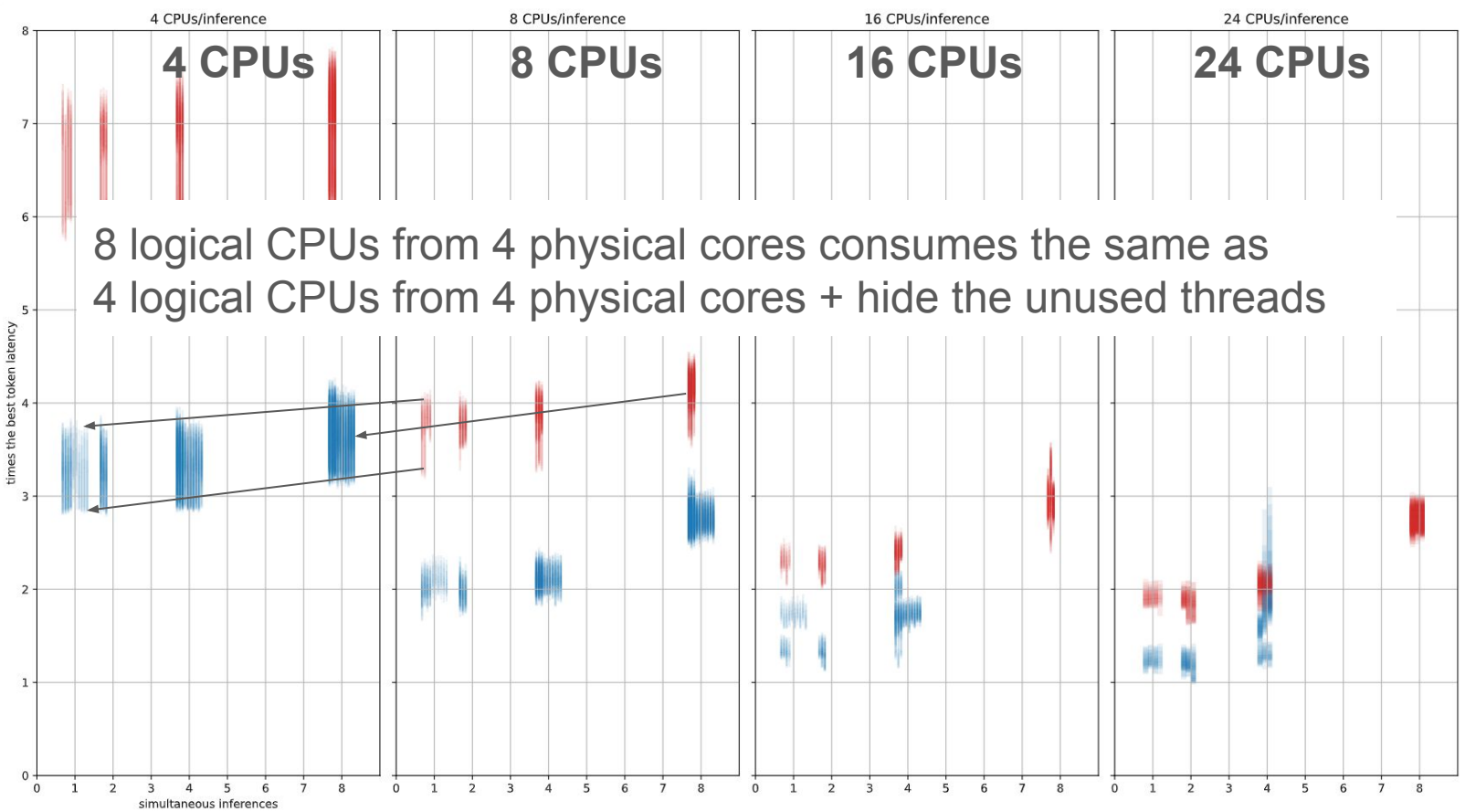
Take logical CPUs from the same physical cores, but use only one.



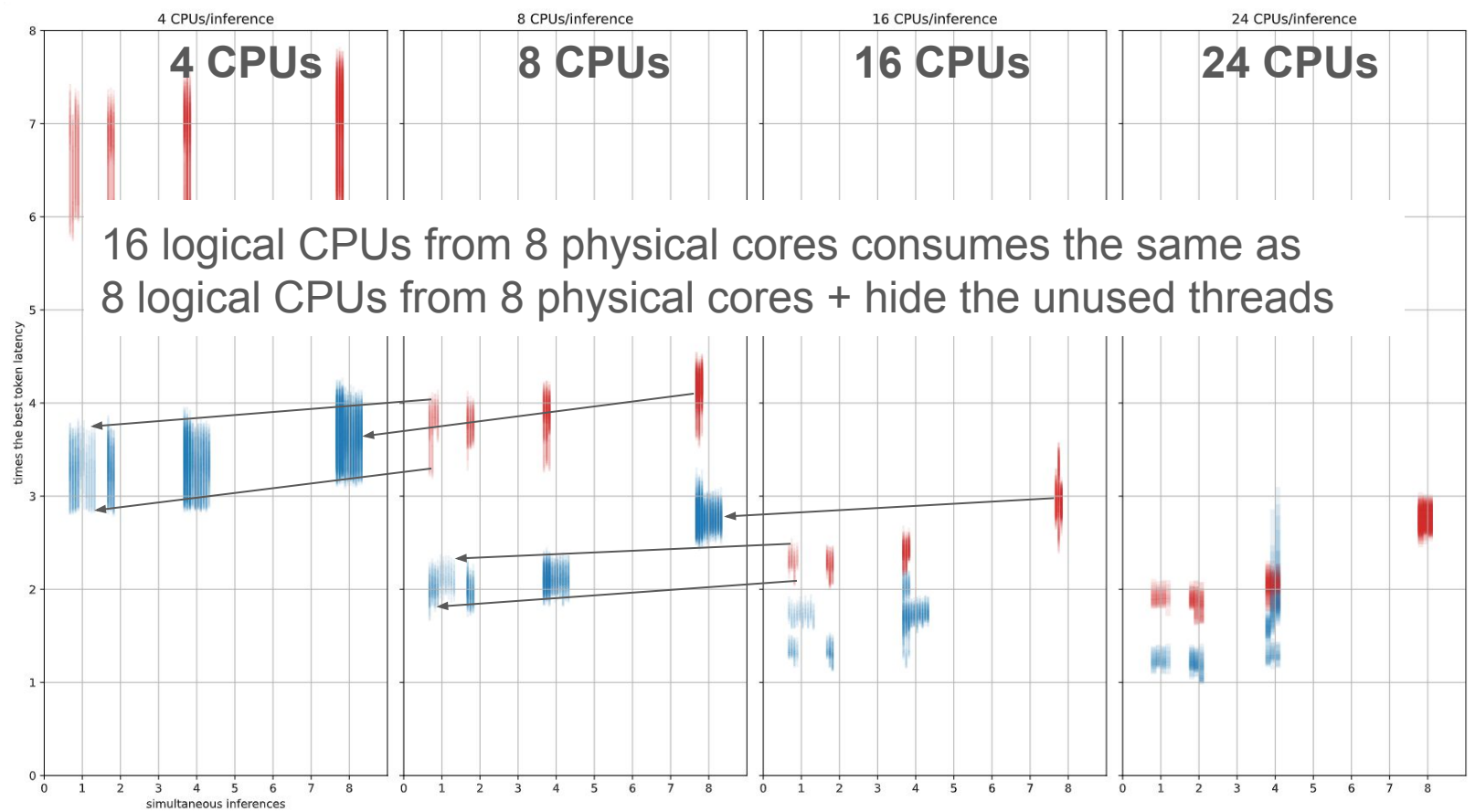
Analyzing 1 or 2 logical CPUs / core



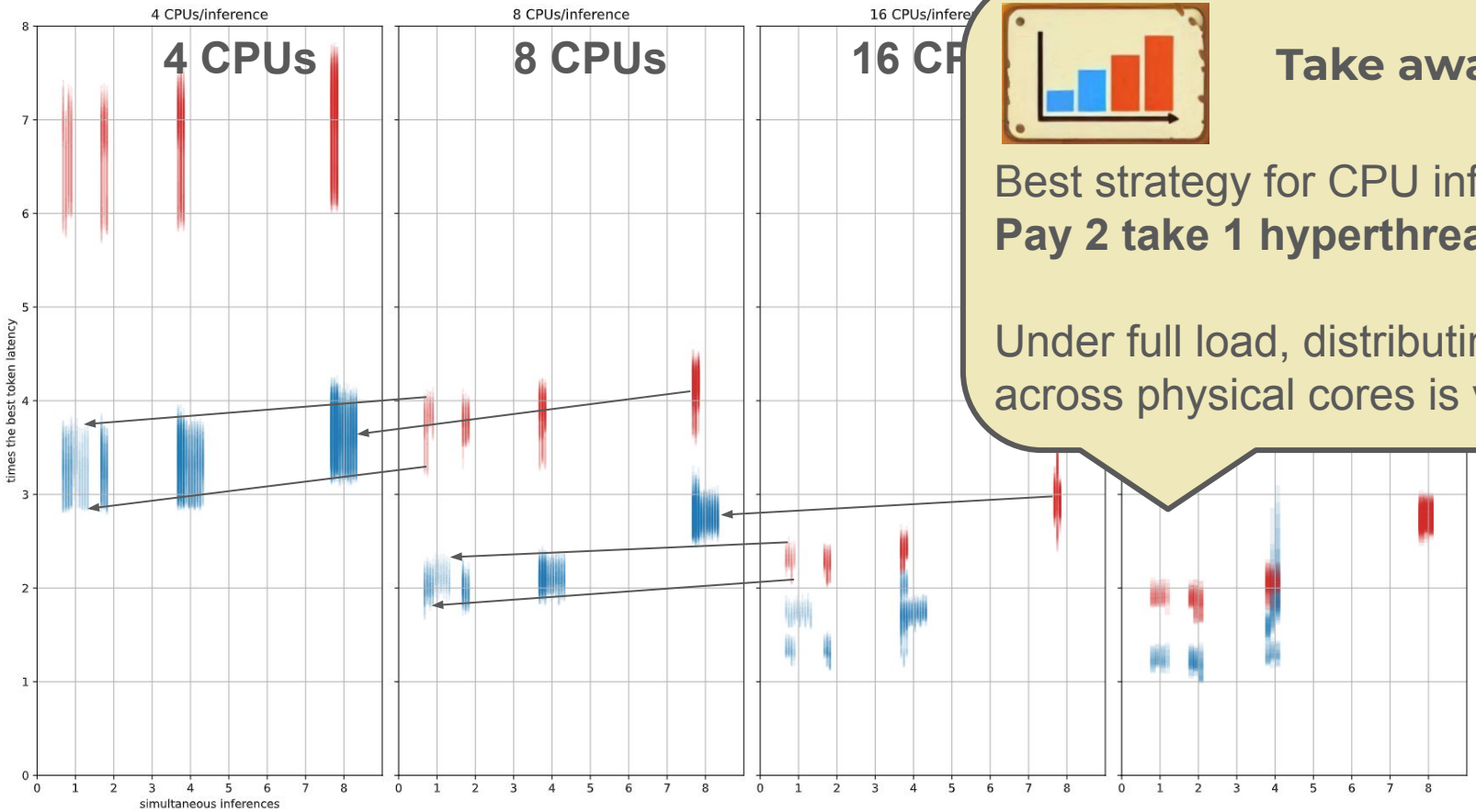
Analyzing 1 or 2 logical CPUs / core



Analyzing 1 or 2 logical CPUs / core



Analyzing 1 or 2 logical CPUs / core

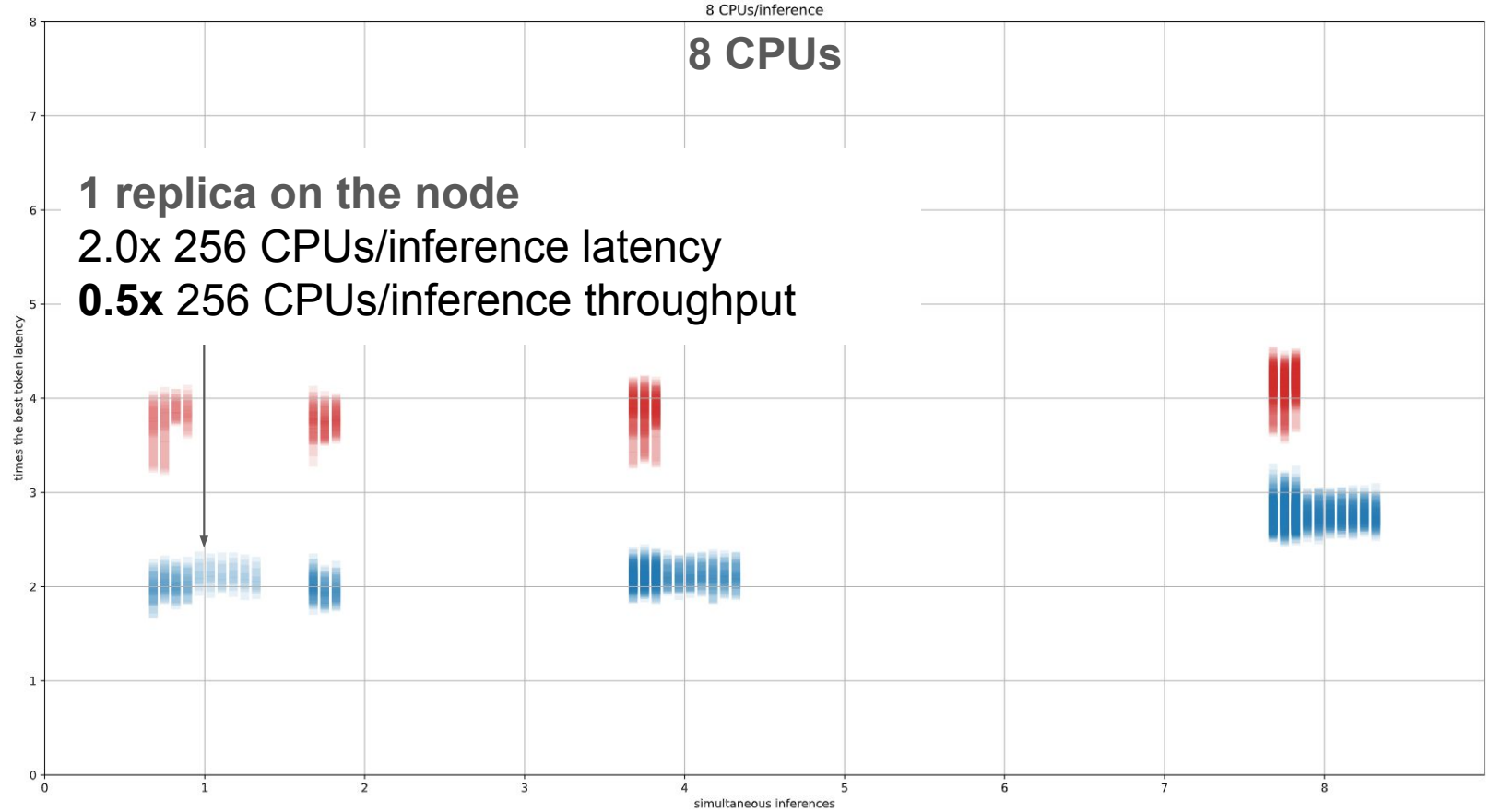


Take away

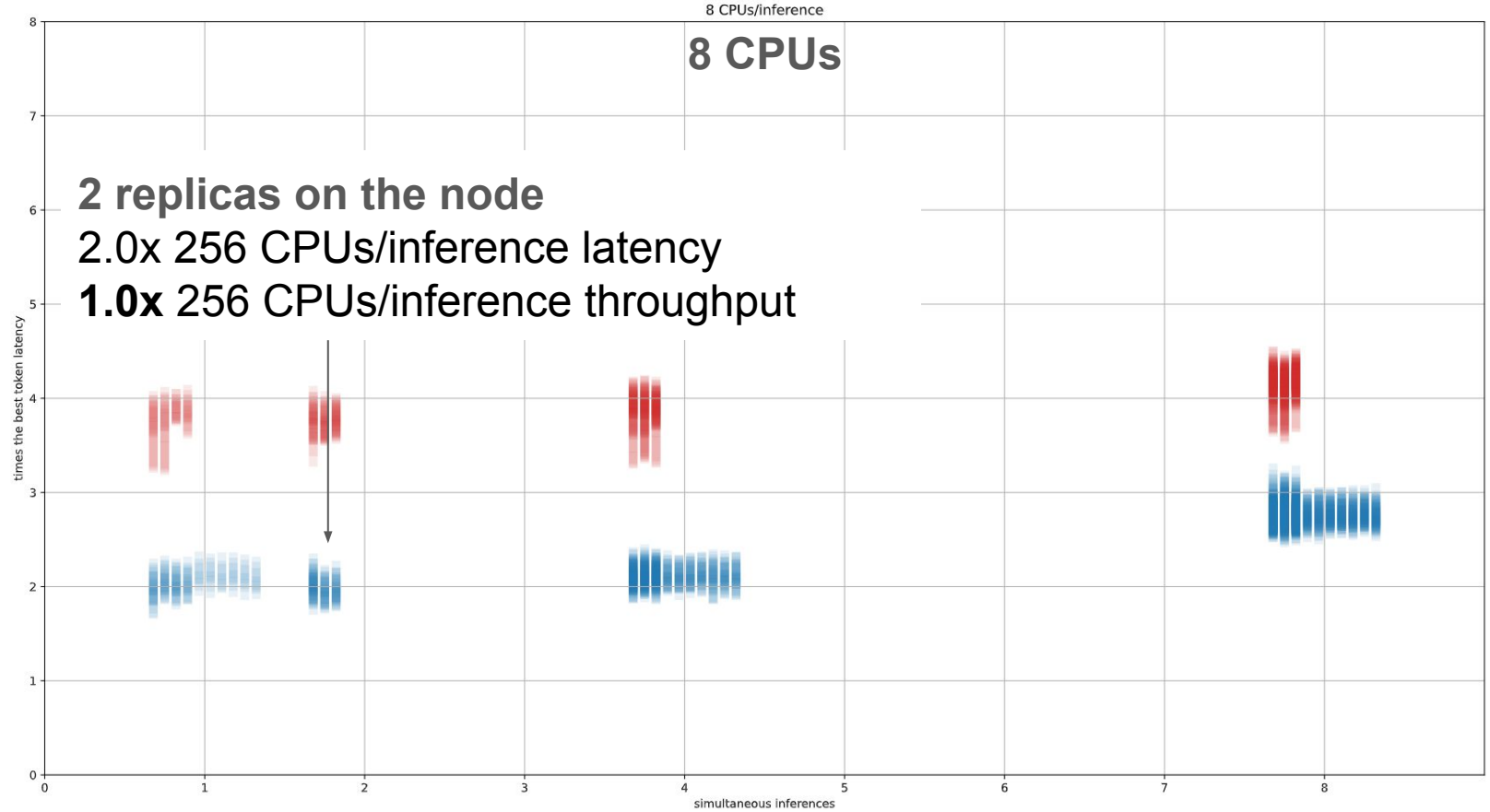
Best strategy for CPU inference:
Pay 2 take 1 hyperthreads.

Under full load, distributing CPUs across physical cores is very bad.

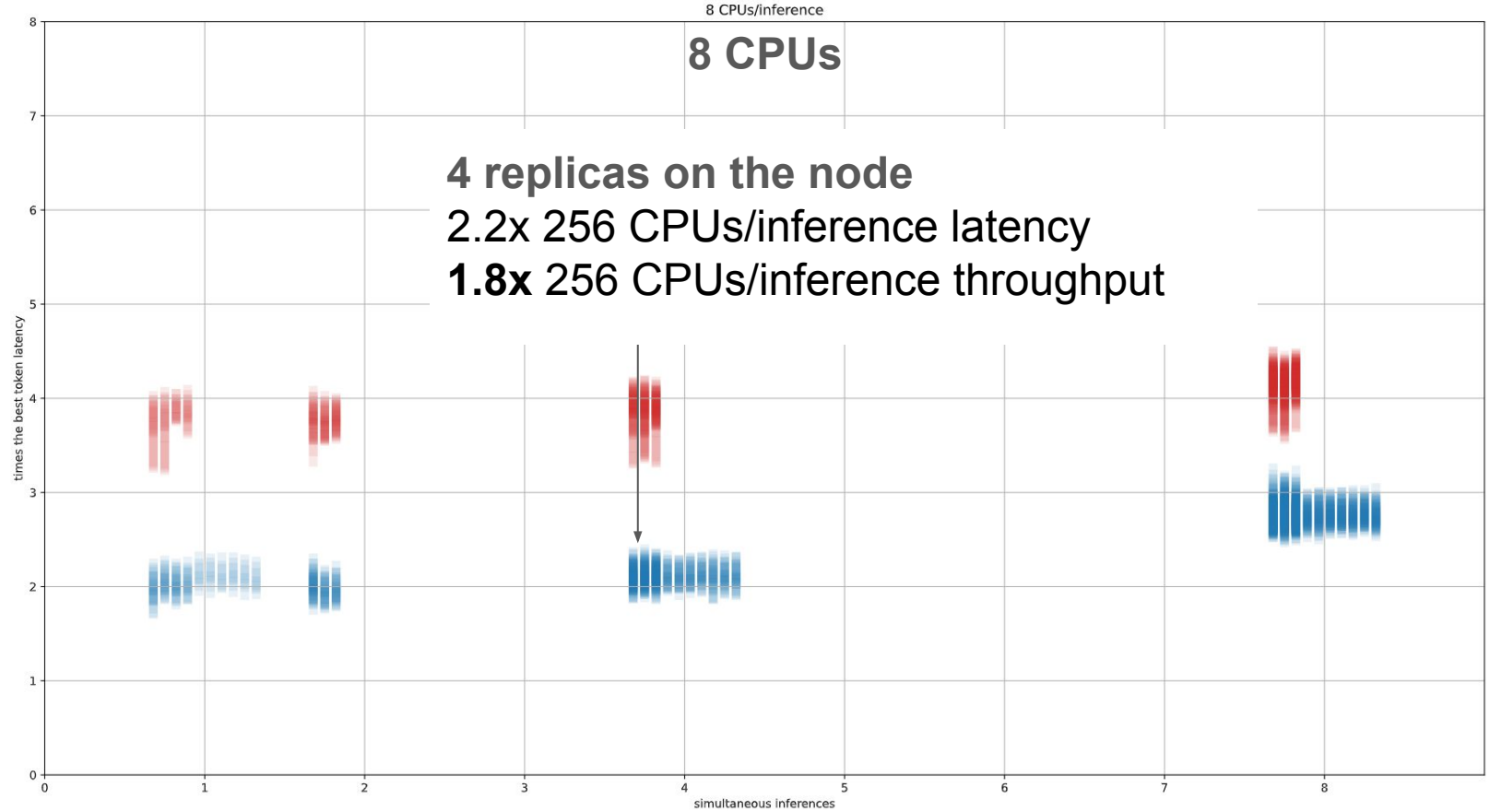
Analyzing node throughput



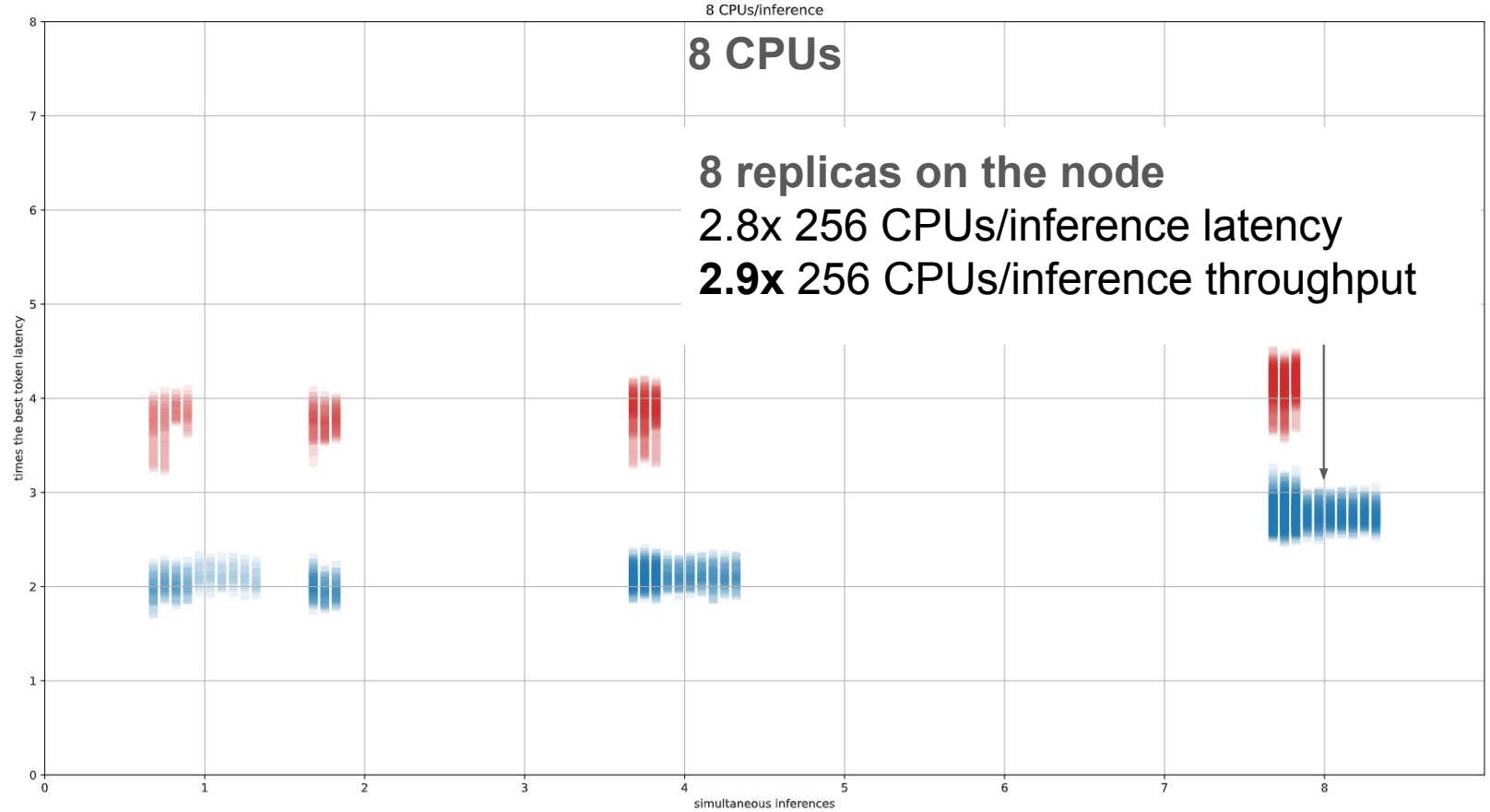
Analyzing node throughput



Analyzing node throughput



Analyzing node throughput



Analyzing node throughput

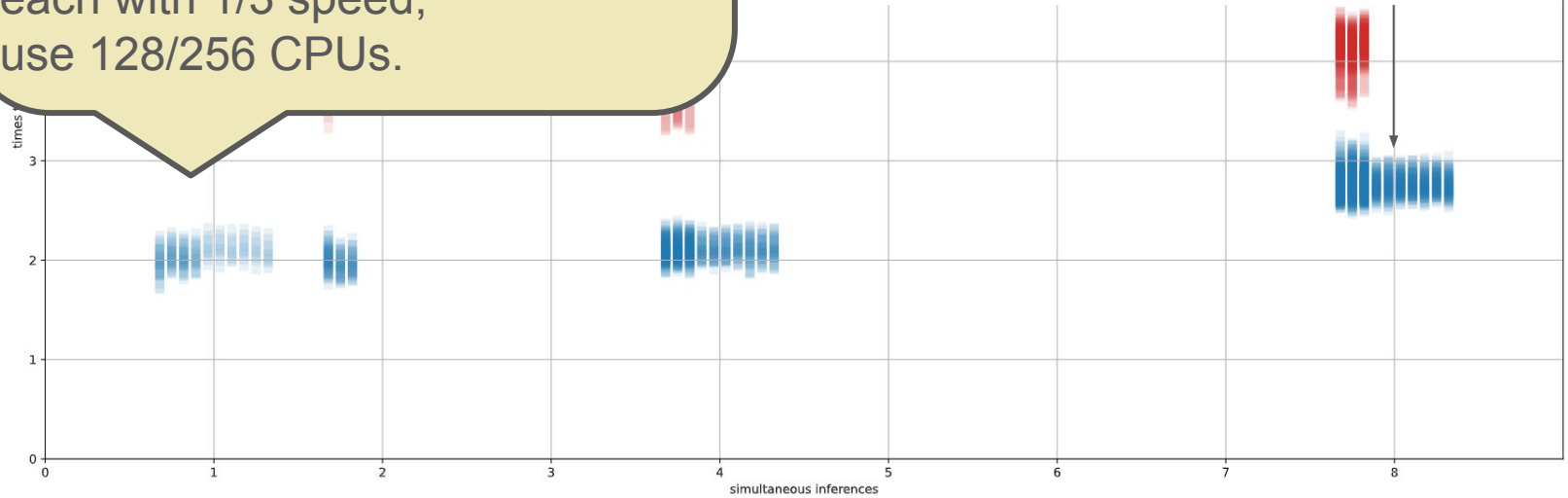
Take away

Balancing latency and throughput:
Triple the server throughput
by running 8 replicas
each with 1/3 speed,
use 128/256 CPUs.

8 CPUs/inference

8 CPUs

8 replicas on the node
2.8x 256 CPUs/inference latency
2.9x 256 CPUs/inference throughput



Take away from methodology

Take away Python

You can transparently override functions in other Python libraries.

Take away utils & examples

CPU/cache/mem metrics: PCM
Full AI solutions: OPEA Examples

Take away data

Store raw timestamped data: freedom to change what you measure/compare later.

Take away resource management

NRI resource policies

- Balloons (configurable)
- Topology-aware (backward compatible k8s semantics, modern hardware support)

Take away from CPU inference

Take away CPU count

Spending 256 CPUs to a single inference is probably not optimal.

Take away NUMA

Sub-NUMA clustering and
1 inference / NUMA node:
Protect inferences from interference.

Take away hyperthreads

Best strategy for CPU inference:
Pay 2 take 1 hyperthreads.

Under full load distributing CPUs across physical cores is very bad.

Take away balance

Balancing latency and throughput:
Triple the server throughput
by running 8 replicas
with 1/3 speed, use 128/256 CPUs.

Thank you so much!


Python Instrument


Raw data


Utils & Examples


NRI resource management

QR code for feedback

Links to demos/examples:

- Instrument
- Resource management

External links:

- OPEA project
- PCM tool
- NRI plugins
- OPEA resource policy doc


CPU count


Sub-NUMA


Hyperthreads


Balance