CopyCat:

Harvesting the Frequency Tax of Bulk Memory Copy

Zecheng Li / Jian Zhang

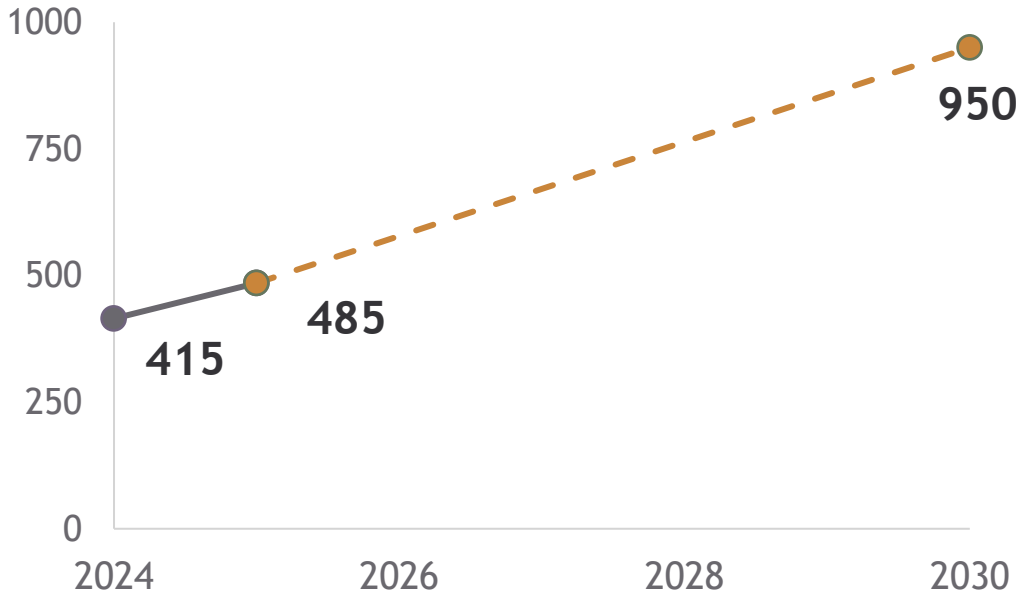
The Hong Kong University of Science and Technology (Guangzhou)

HotStorage ’26

Memory copying is a CPU energy target

Data-center electricity use

TWh per year



Dashed line: 2030 projection

CPUs are a major power consumer

61% of peak power
in a reference data-center model

2017-generation servers · Barroso et al.

Memory copying consumes CPU cycles

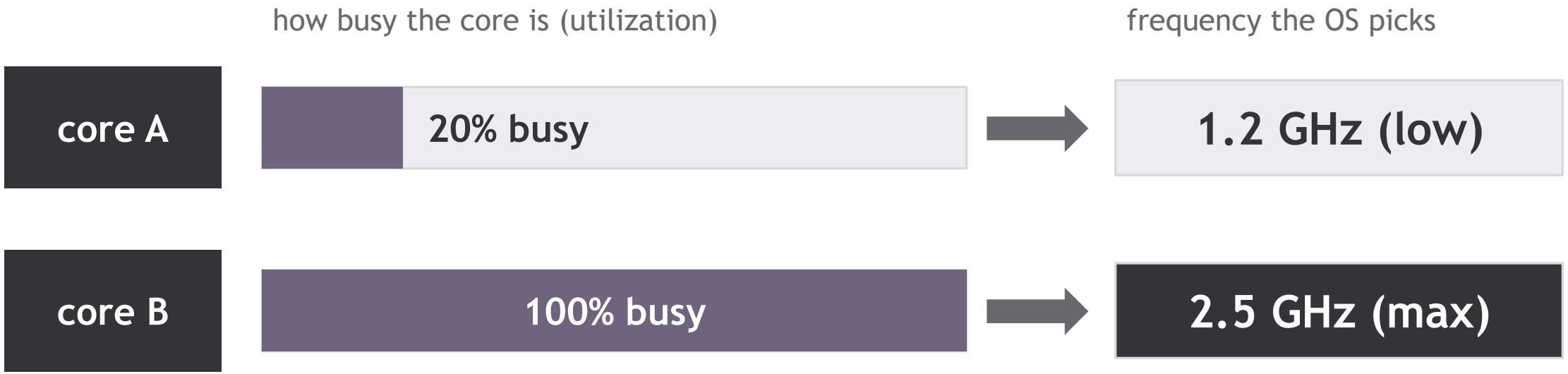
1st of CPU cycles in Google's profile
memcpy + memmove

Fleet study · Kanev et al., ISCA 2015

Our target: reduce the CPU energy spent on memcpy

IEA (2025, 2026); Barroso et al. (2018), Fig. 1.8; Kanev et al., ISCA 2015

Today, each core's frequency follows how busy that core is



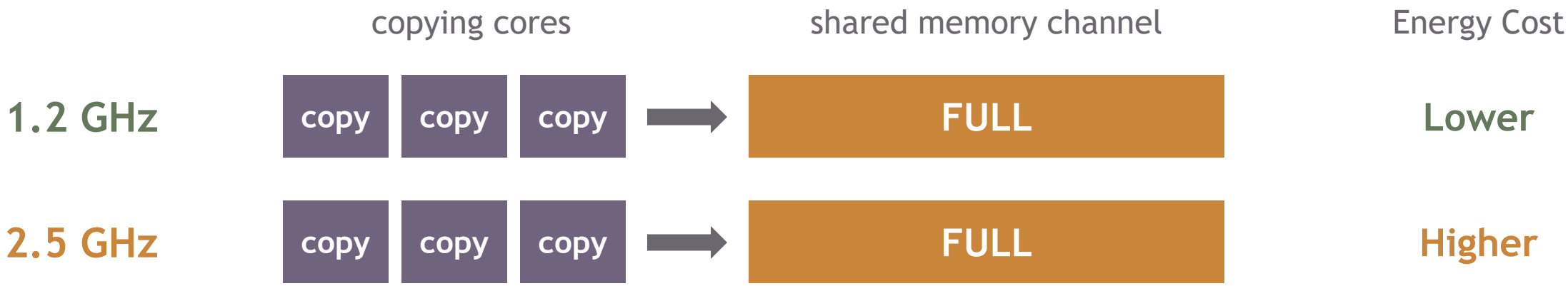
Rule: busy → higher frequency · idle → lower frequency

Linux governor (schedutil) + Intel hardware control (HWP): both decide core by core

Good for computation: a busy core usually **needs** the speed.

HWP (Hardware-managed P-states) lets hardware choose performance within OS (operating system) limits.
schedutil is a software frequency governor using scheduler utilization; the scheduler assigns CPU time.

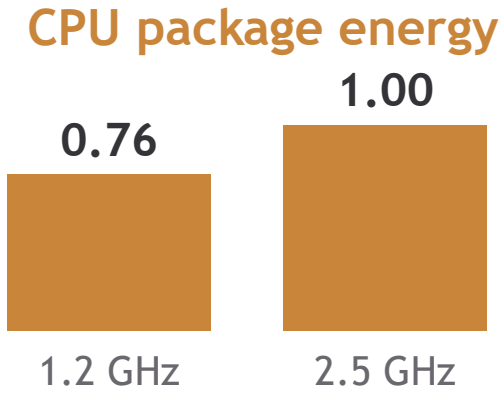
Low-frequency cores already saturate memory bandwidth



Higher frequency: no more throughput gain, more energy cost for the same work

Measured multi-threaded copies in local DRAM





Same throughput

24% less energy

at 1.2 GHz vs. 2.5 GHz

Top: schematic. Bottom: CopyCat §2.2, Fig. 1 (multi-thread case); CPU package energy (RAPL).

We call this waste the

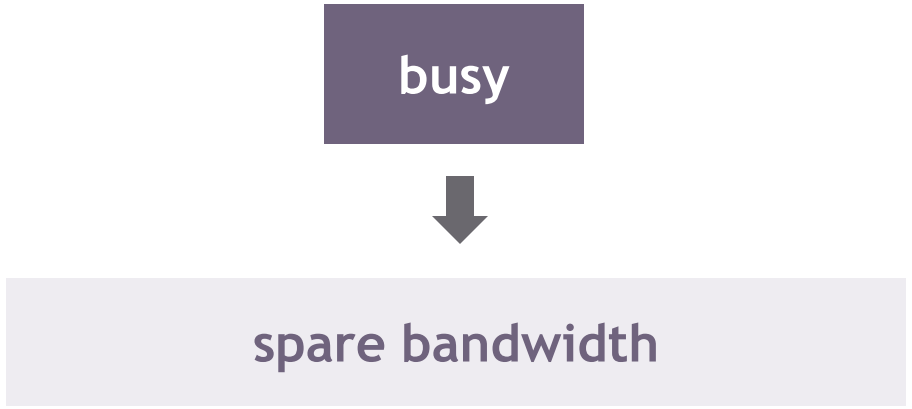
frequency tax

energy paid for a **high frequency**
that buys **no extra throughput**

Problem 1: per-core control misses shared bandwidth limits

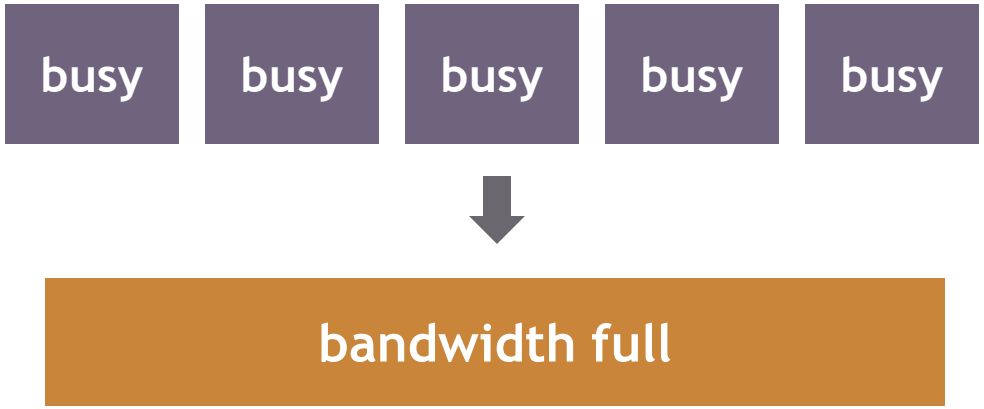
Per-core control chooses each core's frequency from its own activity

one copying thread



lower frequency: copy slows

many copying threads

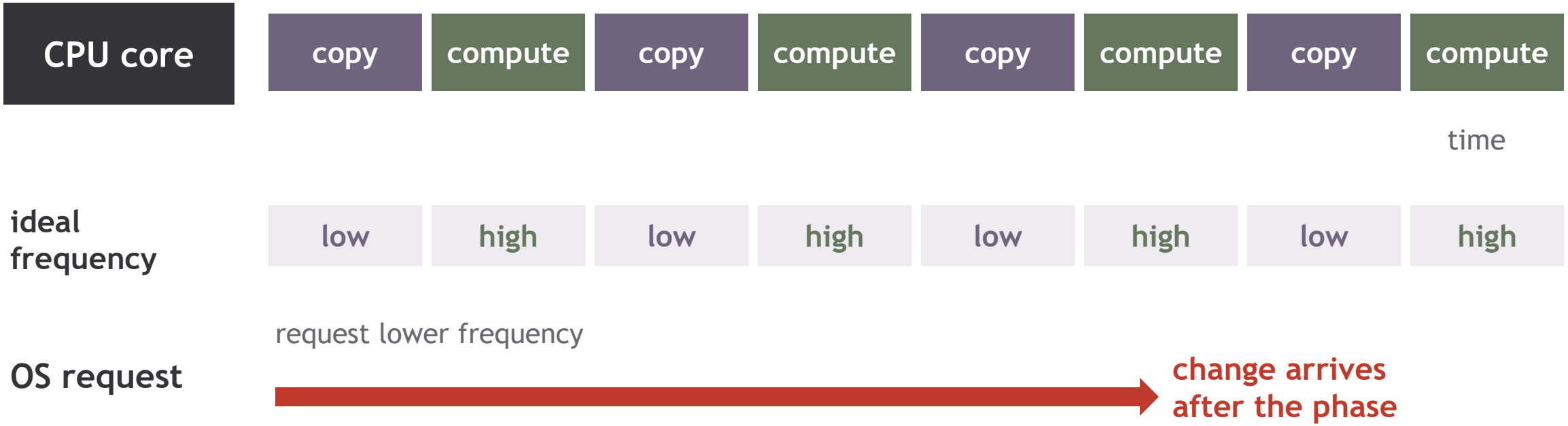


lower frequency: copy holds

Per-core activity does not reveal the shared bandwidth limit

Problem 2: frequency changes lag behind work phases changes

A work item has multiple phases: memory access and computation



Slow frequency changes prevent effective temporal separation of computation and memory access

The main idea behind CopyCat

Make frequency control aware of memory copying

Use aggregate copy throughput to guide frequency

Separate copy and compute across CPU cores

Two design ideas address the two problems

Copy core

Test the throughput response to a lower frequency

Problem 1: observe aggregate copy throughput

App core

Run mixed-workload copies on dedicated cores

Problem 2: place copy and compute on different CPU cores

Continuous copies stay inline; mixed copies can offload

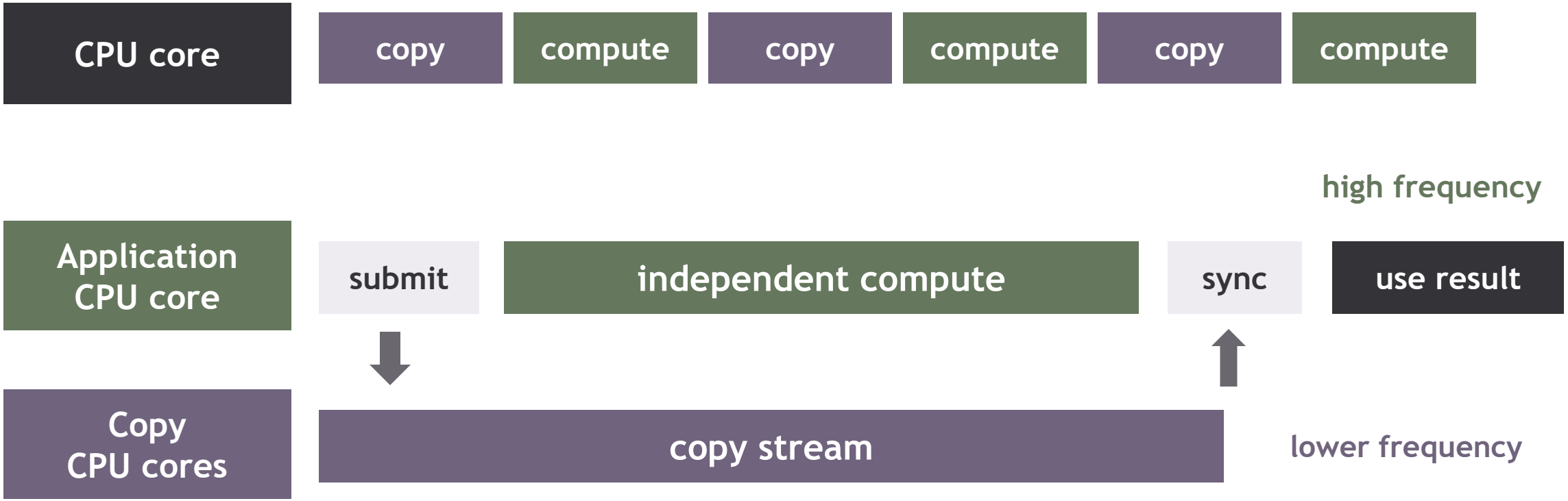
Idea 1: test a lower frequency against copy throughput



Restore the last setting that met the throughput budget

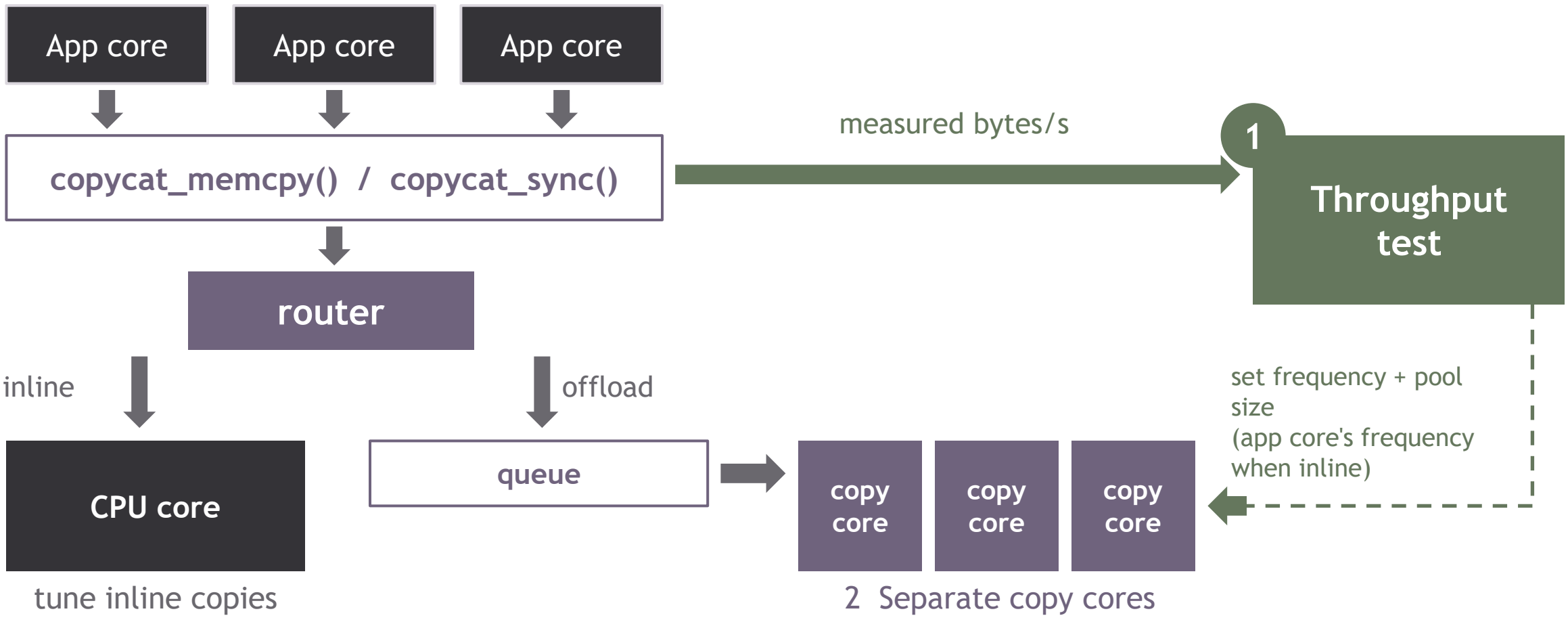
CopyCat · §3.1 · schematic; bar lengths illustrate the decision rule

Idea 2: give copying its own cores



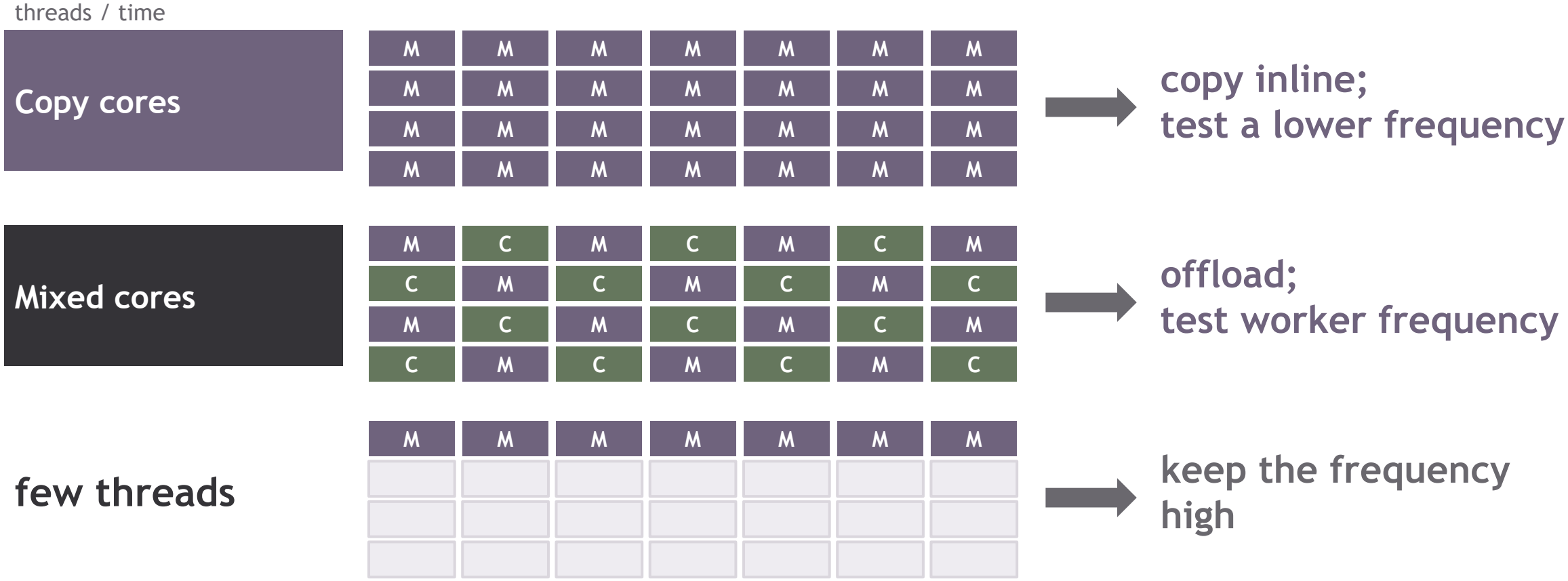
Separate the work, then tune the copy workers

One runtime combines the two ideas



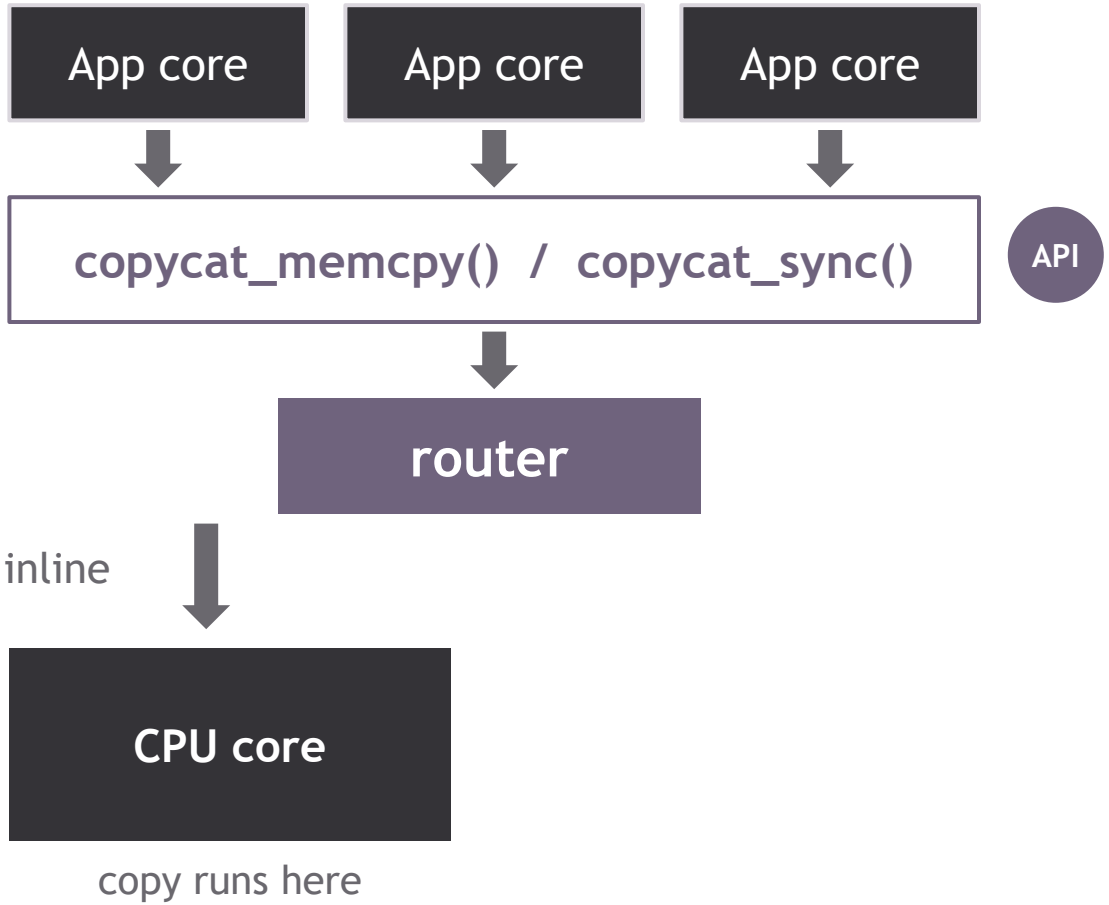
The throughput test guards both inline and offloaded copies

The router selects where copies run



Routing uses the number of submitting threads and their copy activity

The API makes the overlap explicit



Example call site

```
// Submit a copy
copycat_memcpy(dst, src, n);

compute_independent_work();

// Wait before reading dst
copycat_sync();

consume(dst);
```

Submit a copy, do independent work, then wait before using the result

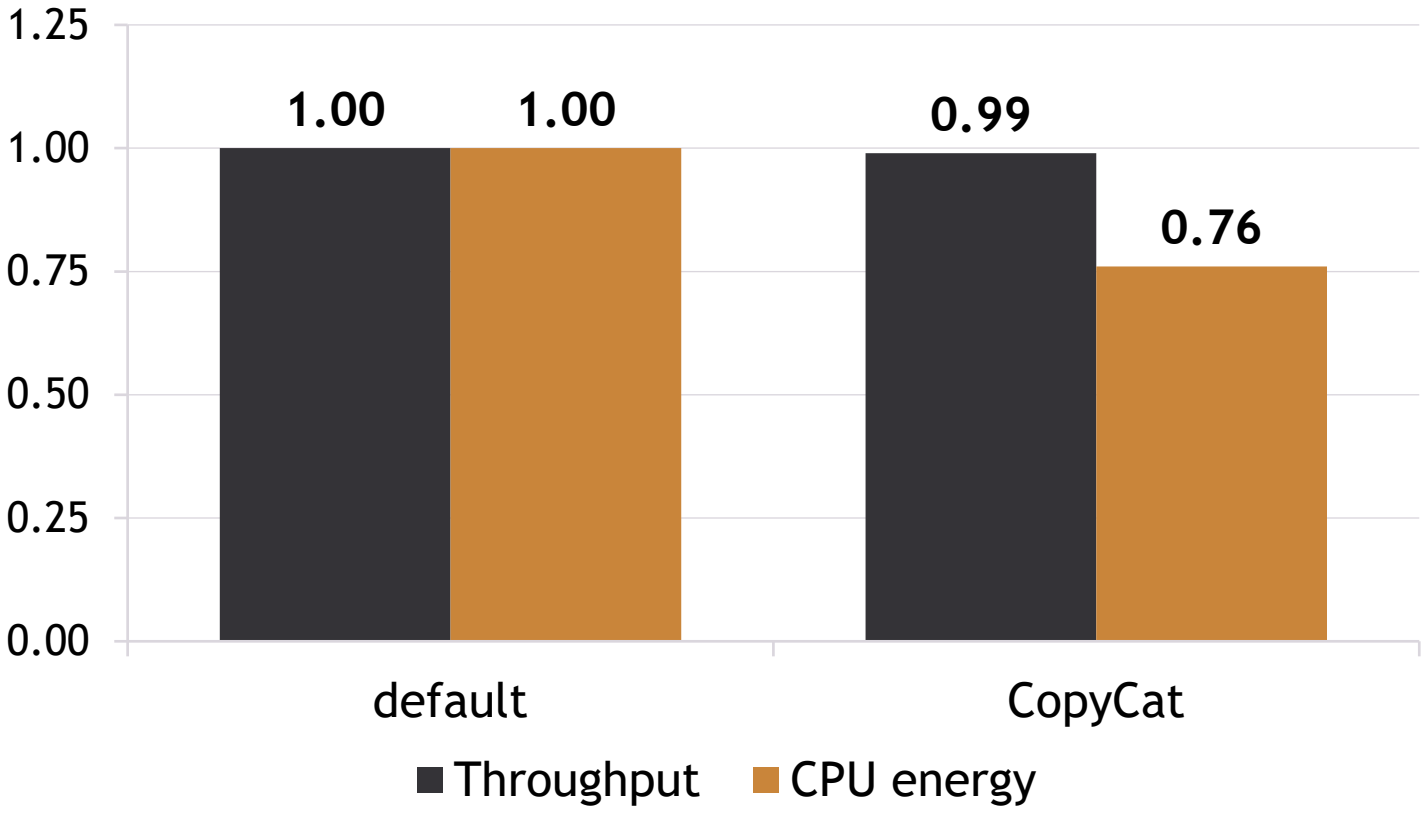
Evaluation: the scope of the measurements

Platform	Dual-socket Intel Xeon Skylake-SP Copies target remote NUMA memory
Comparison	Default Linux schedutil + HWP Application throughput and RAPL package energy Worker energy included

Pure copy **Mixed copy + compute** **Blob-cache prototype**

Pure copying: the controller recovers the saving

Multi-threaded remote-memory writes; no compute phase



−24%

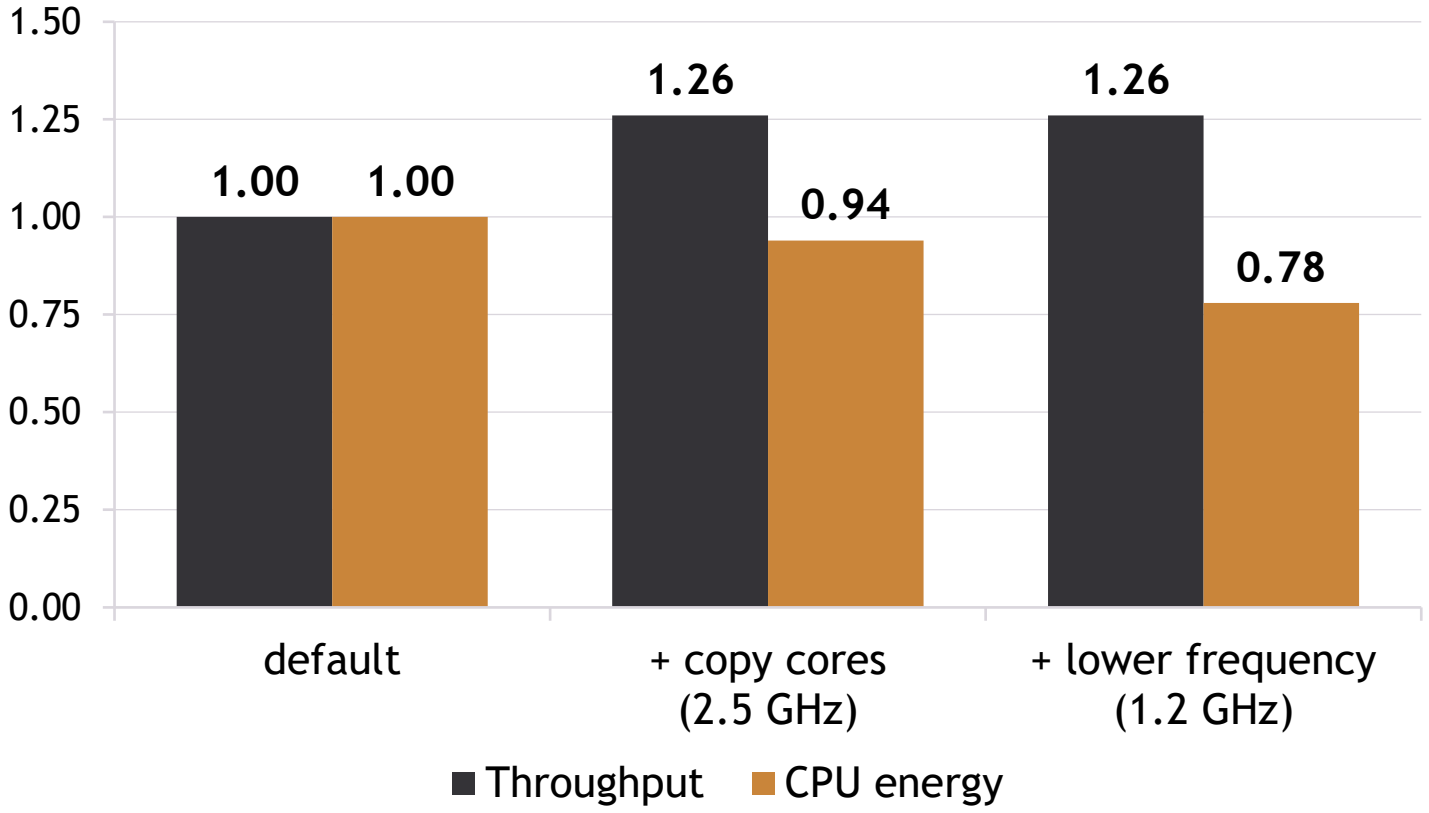
CPU energy

−1% throughput

CopyCat · §4.1, Figure 6(a) · energy = CPU package energy (RAPL)

Mixed work: overlap adds speed, a lower frequency saves energy

4 application threads + 6 copy workers · 1 MiB copies · remote NUMA



+26%

throughput, from overlap

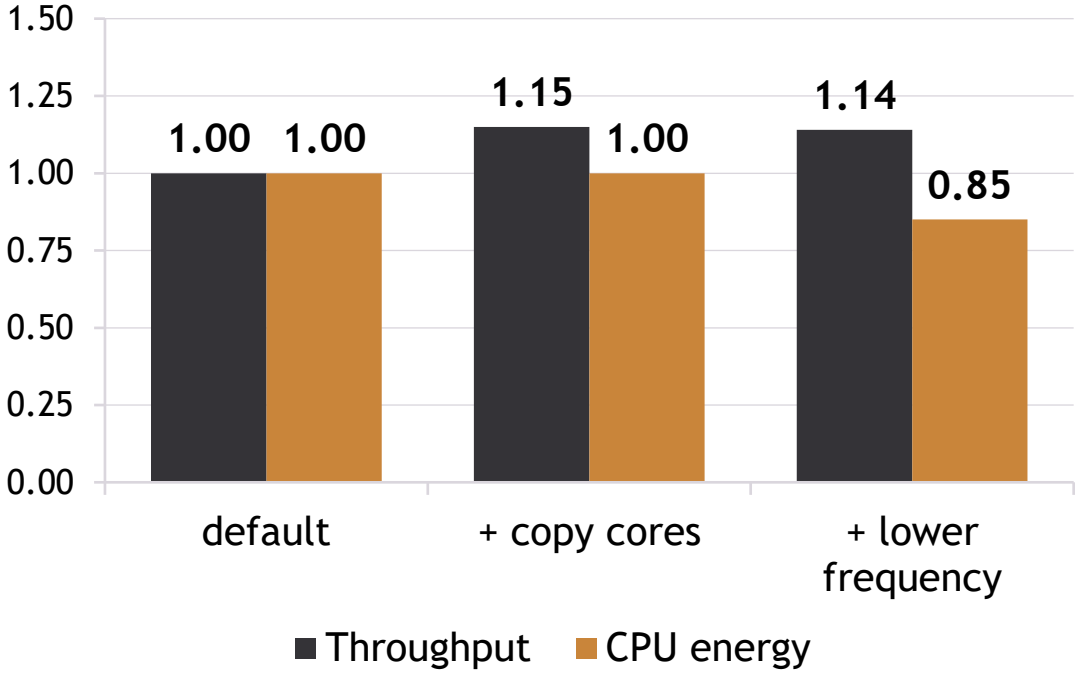
−22%

energy, mostly from the copy cores' lower frequency

CopyCat · §4.2, Figure 6(b) · energy = CPU package energy (RAPL)

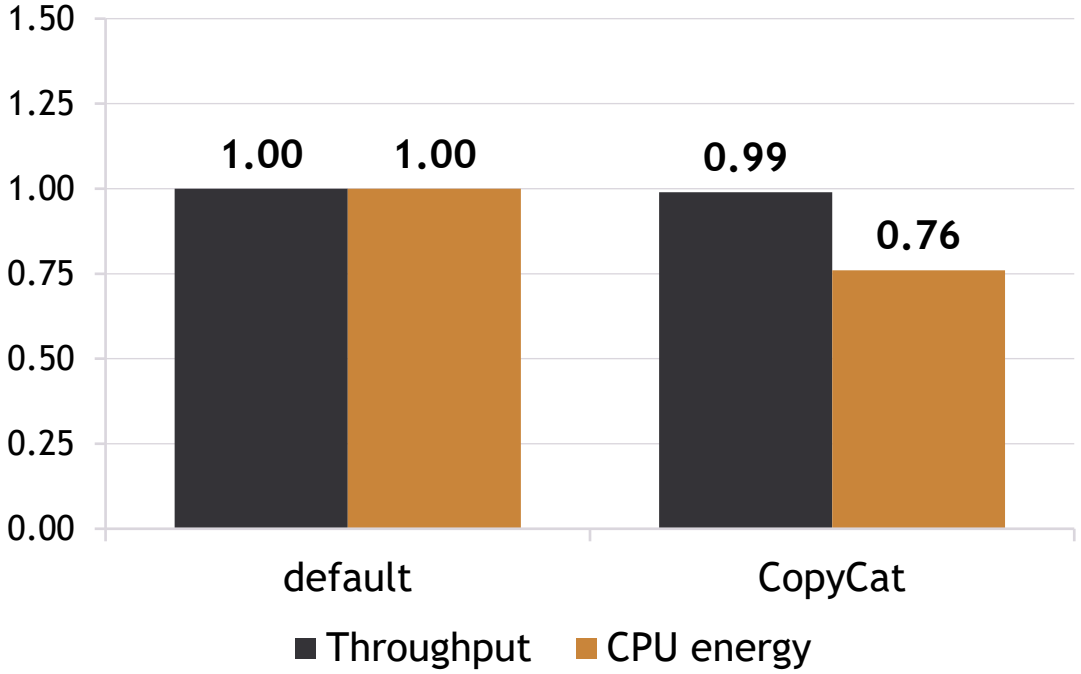
A blob-cache prototype uses both execution paths

Serve: 5 handlers + 5 copy workers



+14% throughput **−15%** energy

Compact: Multiple threads, copies stay inline



−1% throughput **−24%** energy

copy + compute → copy cores · **copy only** → lower frequency in place

CopyCat · §4.3, Figure 6(c)(d) · energy = CPU package energy (RAPL)

CopyCat lowers the frequency where copy throughput holds

Full memory bandwidth can leave high-frequency cores waiting

1. **Test a lower frequency against measured copy throughput**
2. **Separate copying from computation when they alternate**

Up to 24% less CPU package energy in our experiments

Thank you · Zecheng Li · zli221@connect.hkust-gz.edu.cn