From architecture to algorithms: Lessons from the FAWN project

David Andersen, Vijay Vasudevan, Michael Kaminsky*, Michael A. Kozuch*, Amar Phanishayee, Lawrence Tan, Jason Franklin, Iulian Moraru, Sang Kil Cha, Hyeontaek Lim, Bin Fan, Reinhard Munz, Nathan Wan, Jack Ferris, Hrishikesh Amur**, Wolfgang Richter, Michael Freedman***, Wyatt Lloyd***, Padmanabhan Pillali*, Dong Zhou

Carnegie Mellon University *Intel Labs Pittsburgh
** Princeton University *** Georgia Tech

Power limits computing

Infrastructure: PUE

2005: 2–3

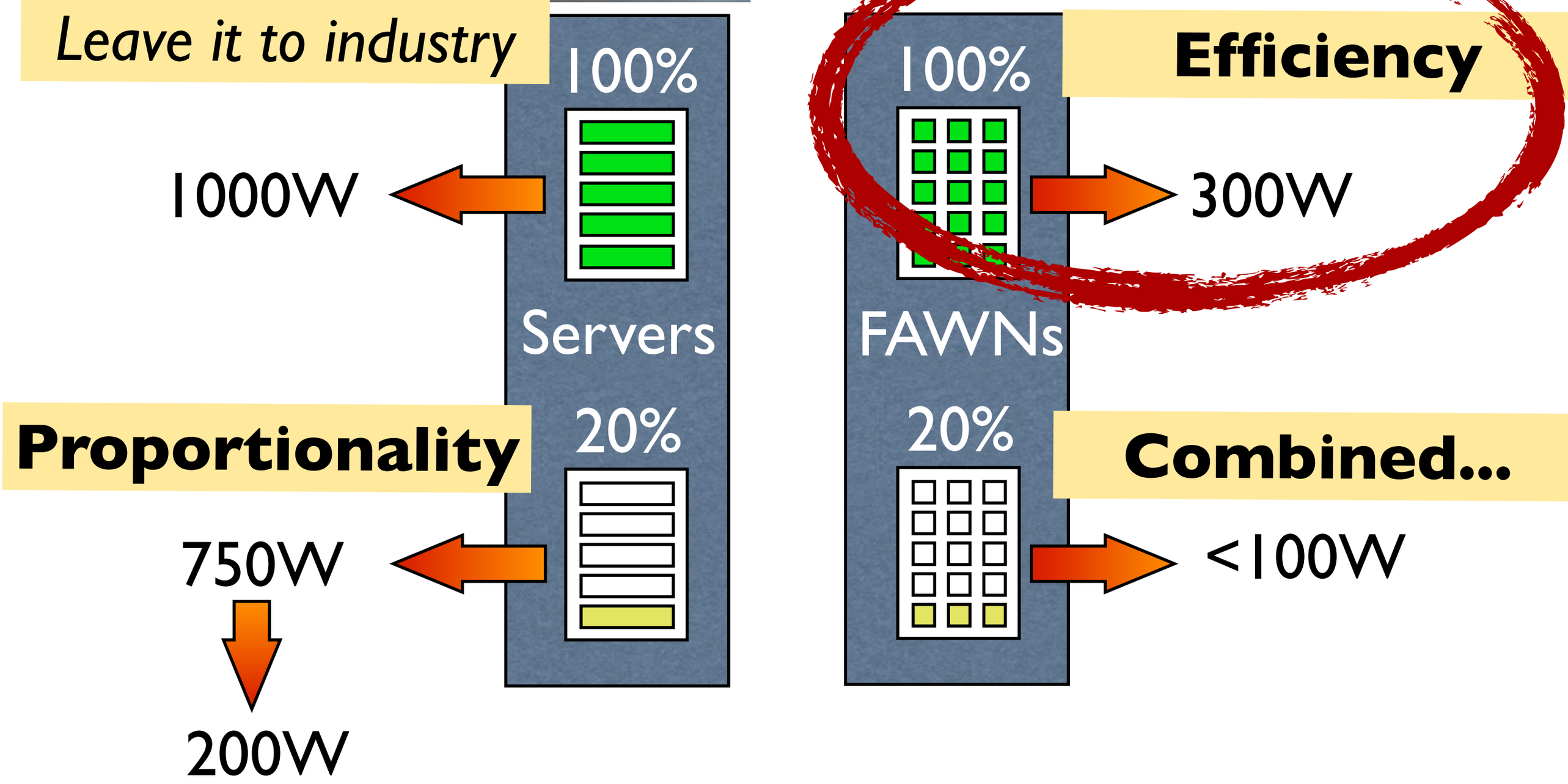
2012: ~1.1

Leave it to industry

Efficiency

Proportionality

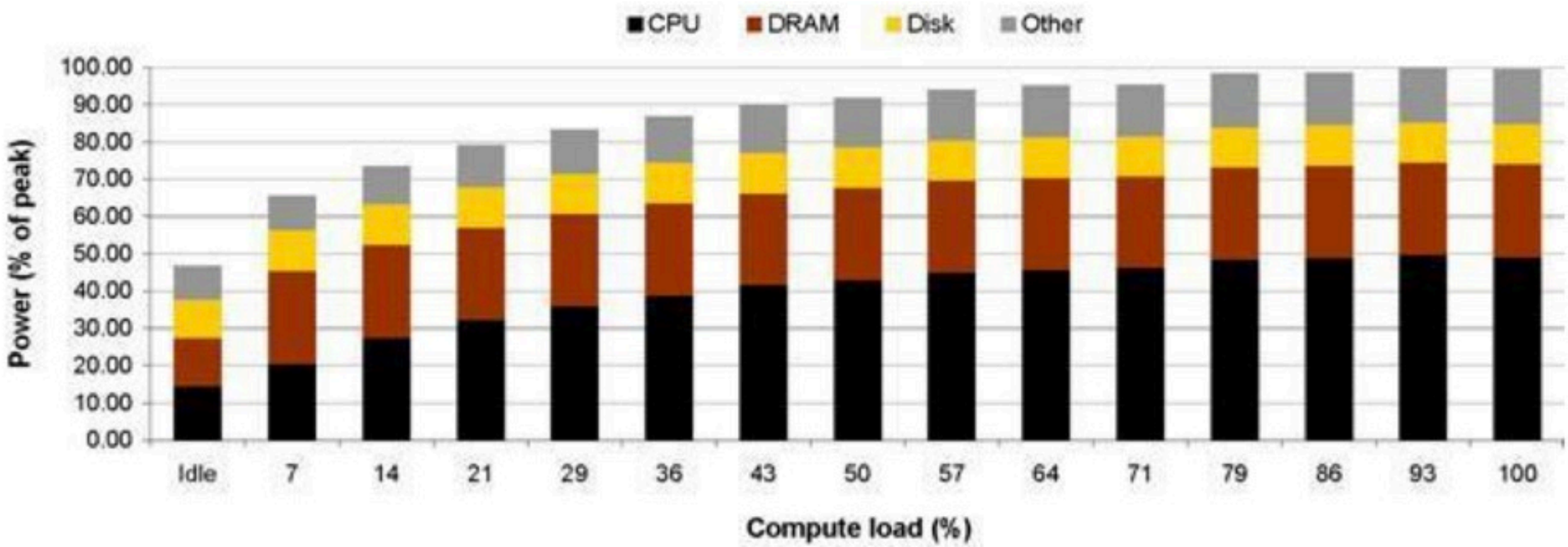
Combined...



Builds up to...

- Computation / total energy =
  1/PUE * 1/SPUE * Computational Work/
  Total Energy to Components

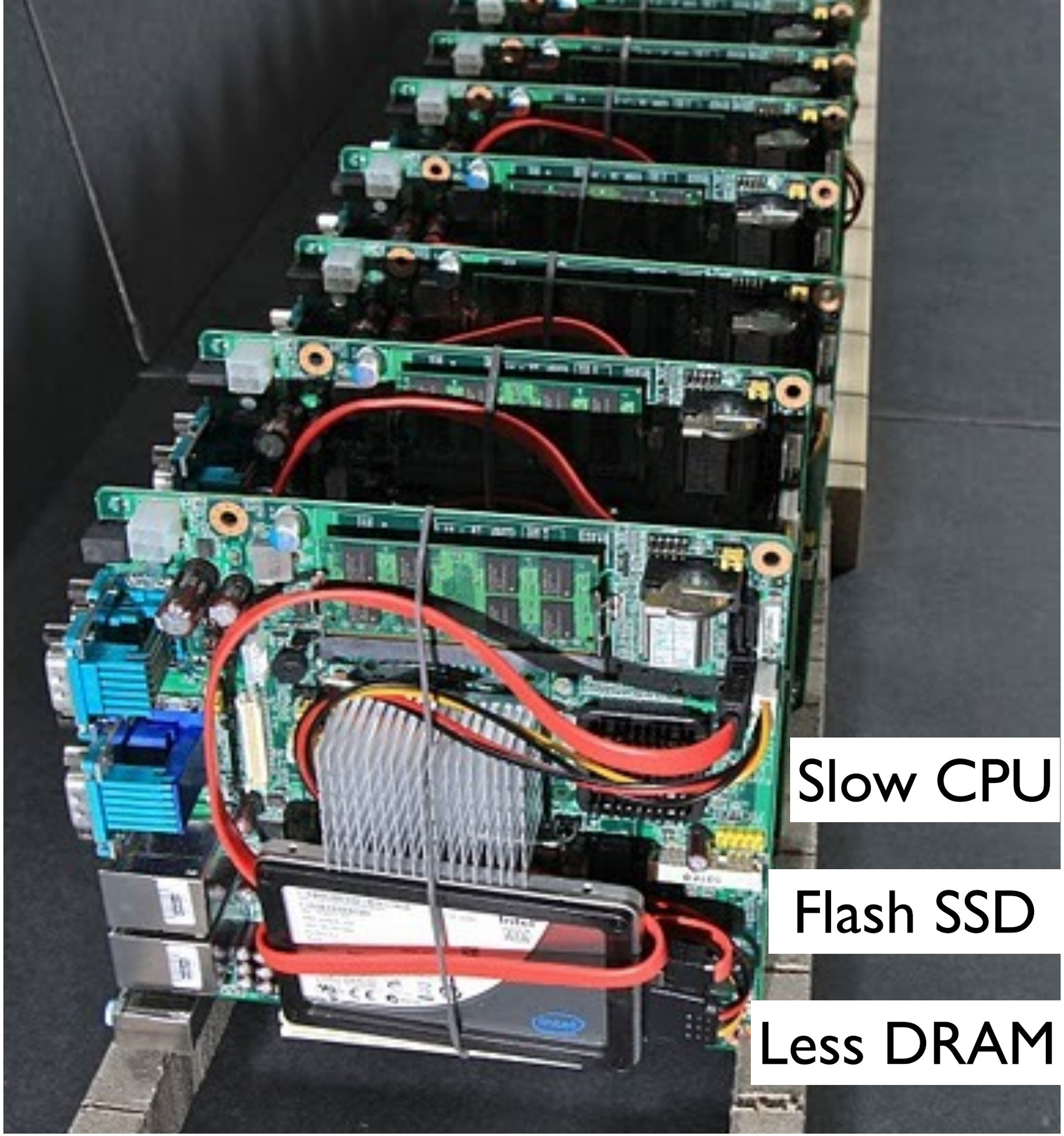
Efficiency vs load



Source: WSC book

Points to make

- DVFS used to be effective; not so good anymore
- Core is decently proportional; rest of components aren't.

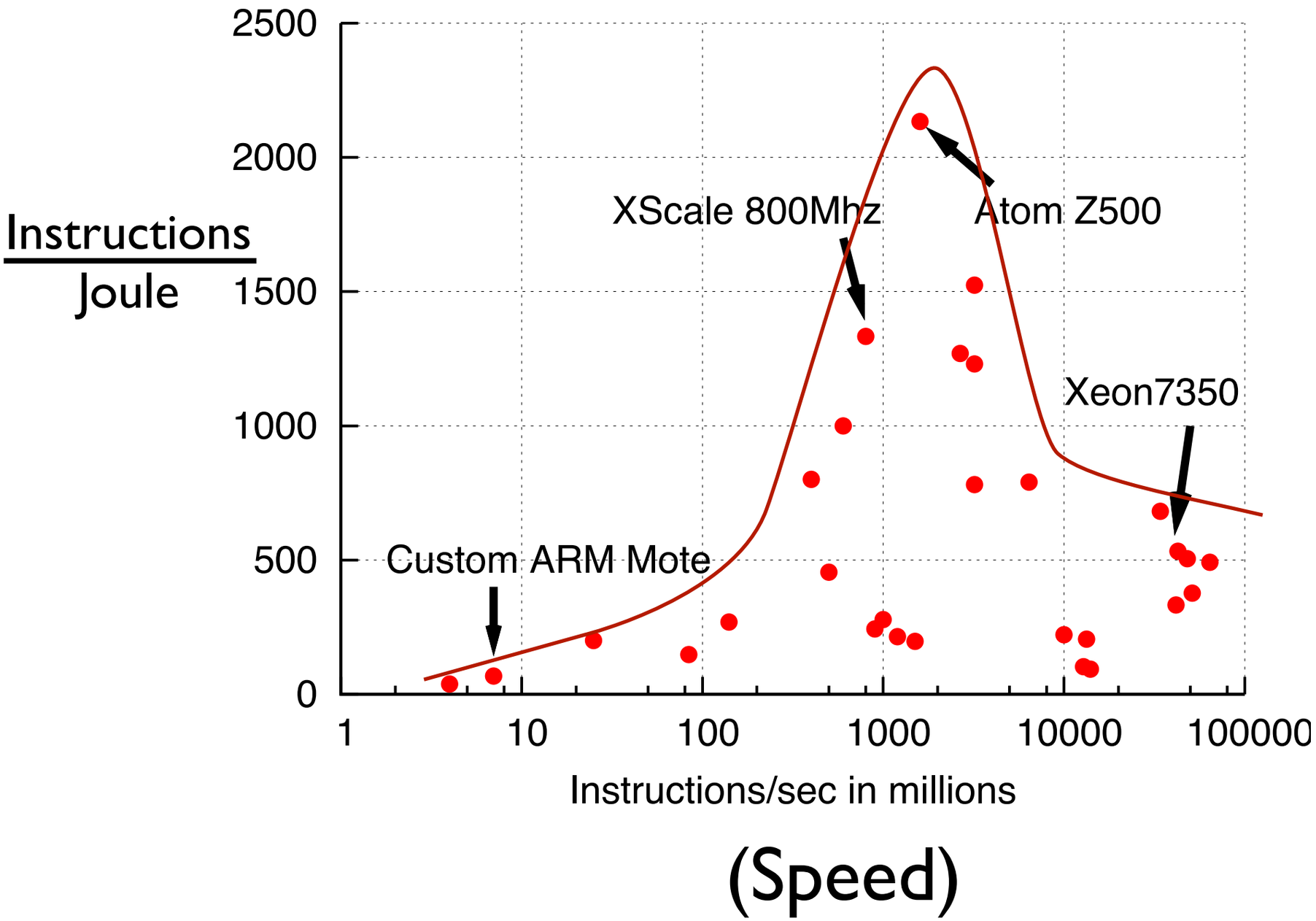

Slow CPU
Flash SSD
Less DRAM

Efficiency is not free

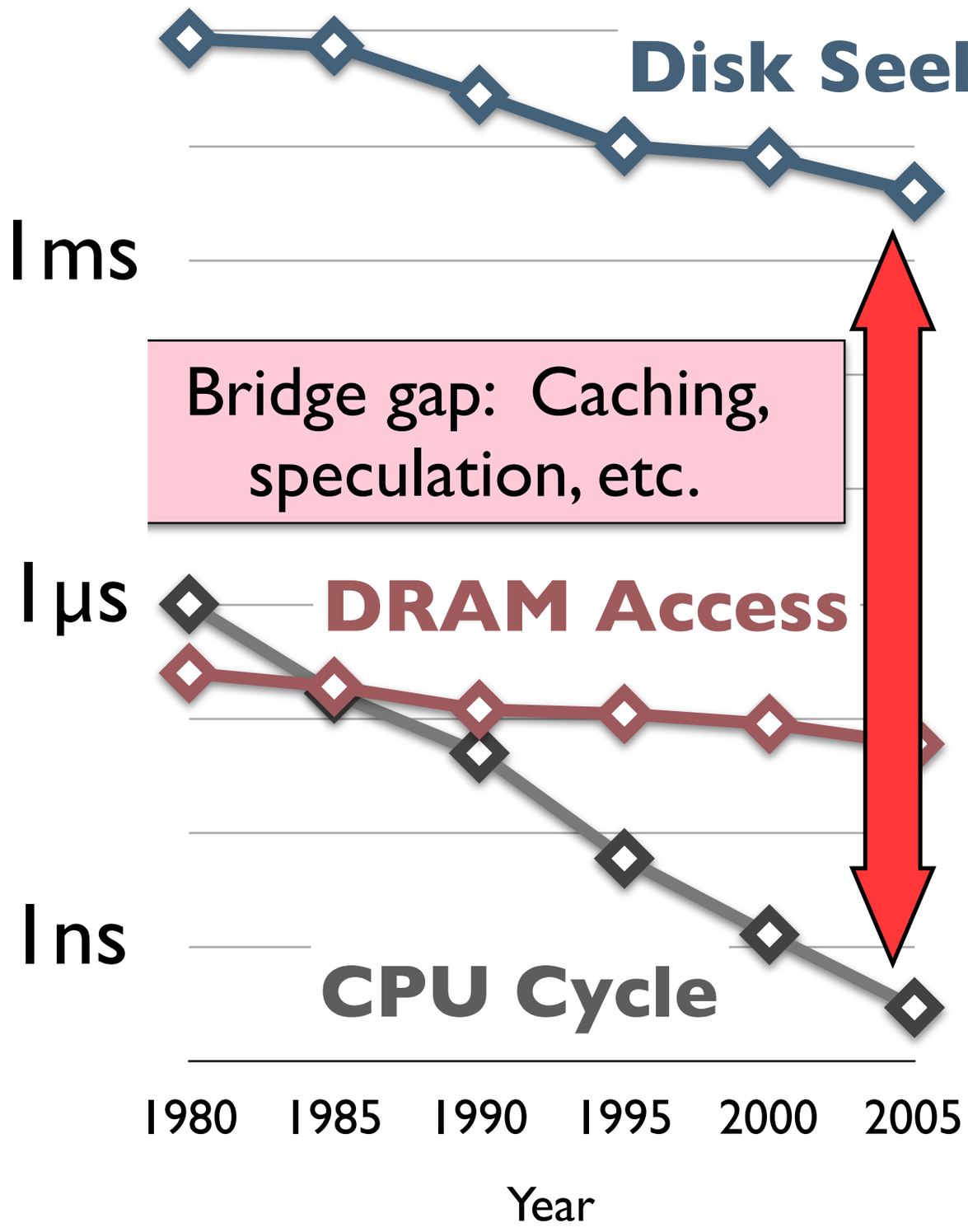
Because speed is not cheap

Gigahertz is not free

Speed and power calculated from specification sheets
Power includes "system overhead" (e.g., Ethernet)

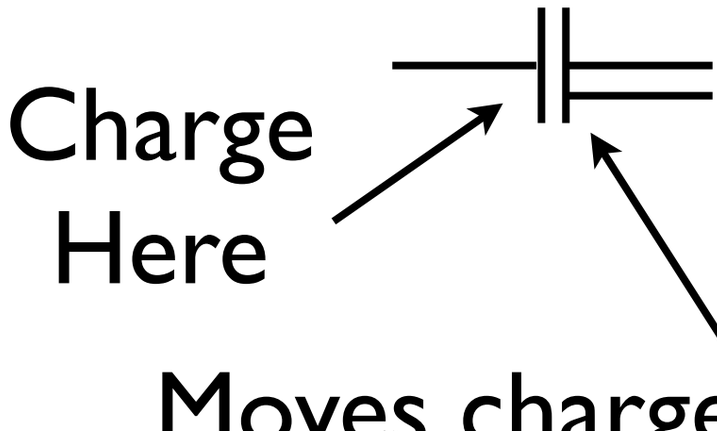


The Memory Wall



Transistors

Have the soul of a capacitor

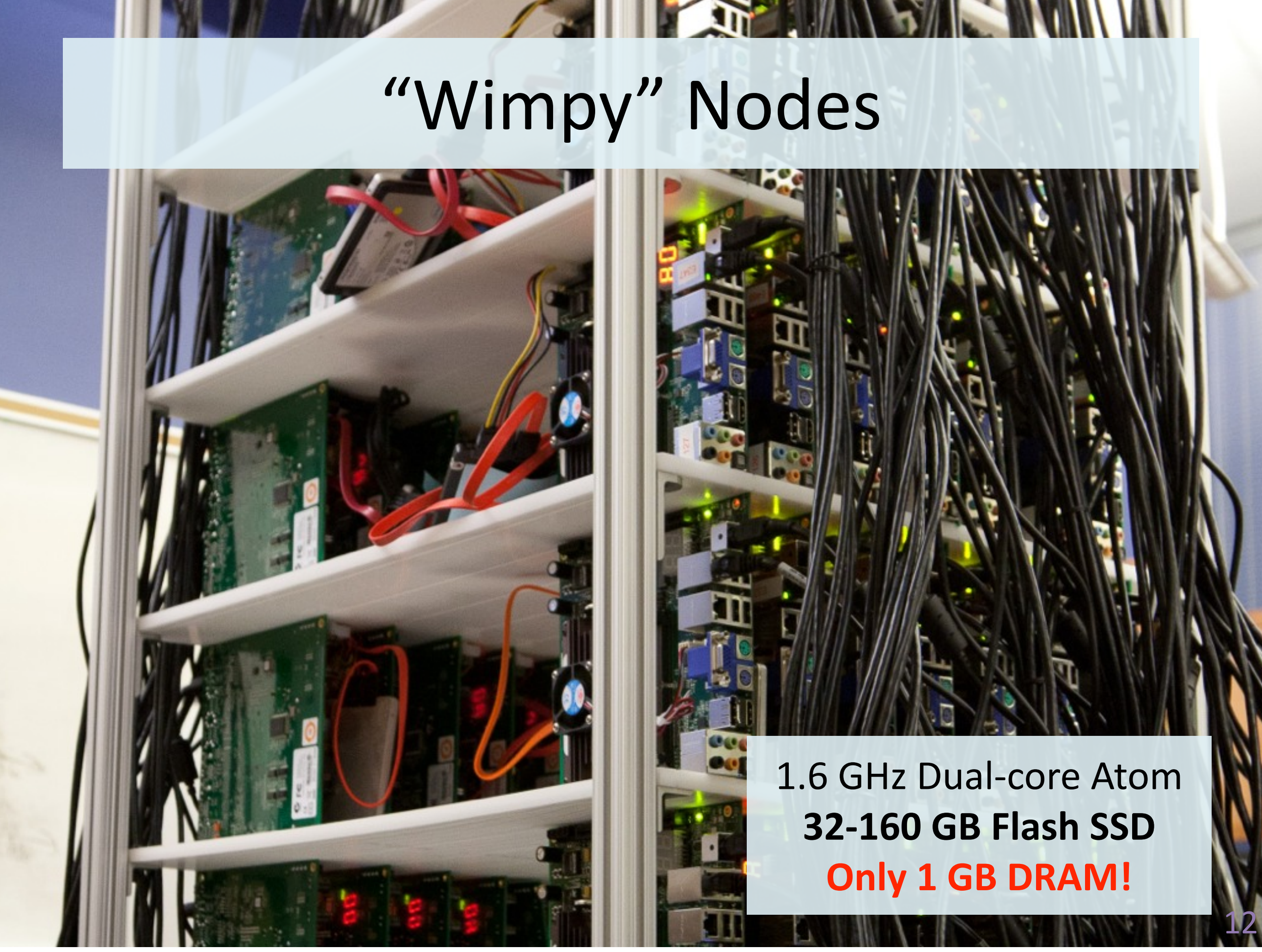


Which lets current flow

Two pillars

- Gigahertz costs twice:
  - Once for the switching speed
  - Once for the memory wall
- Memory capacity costs (at least) once:
  - Longer buses < efficient

“Wimpy” Nodes



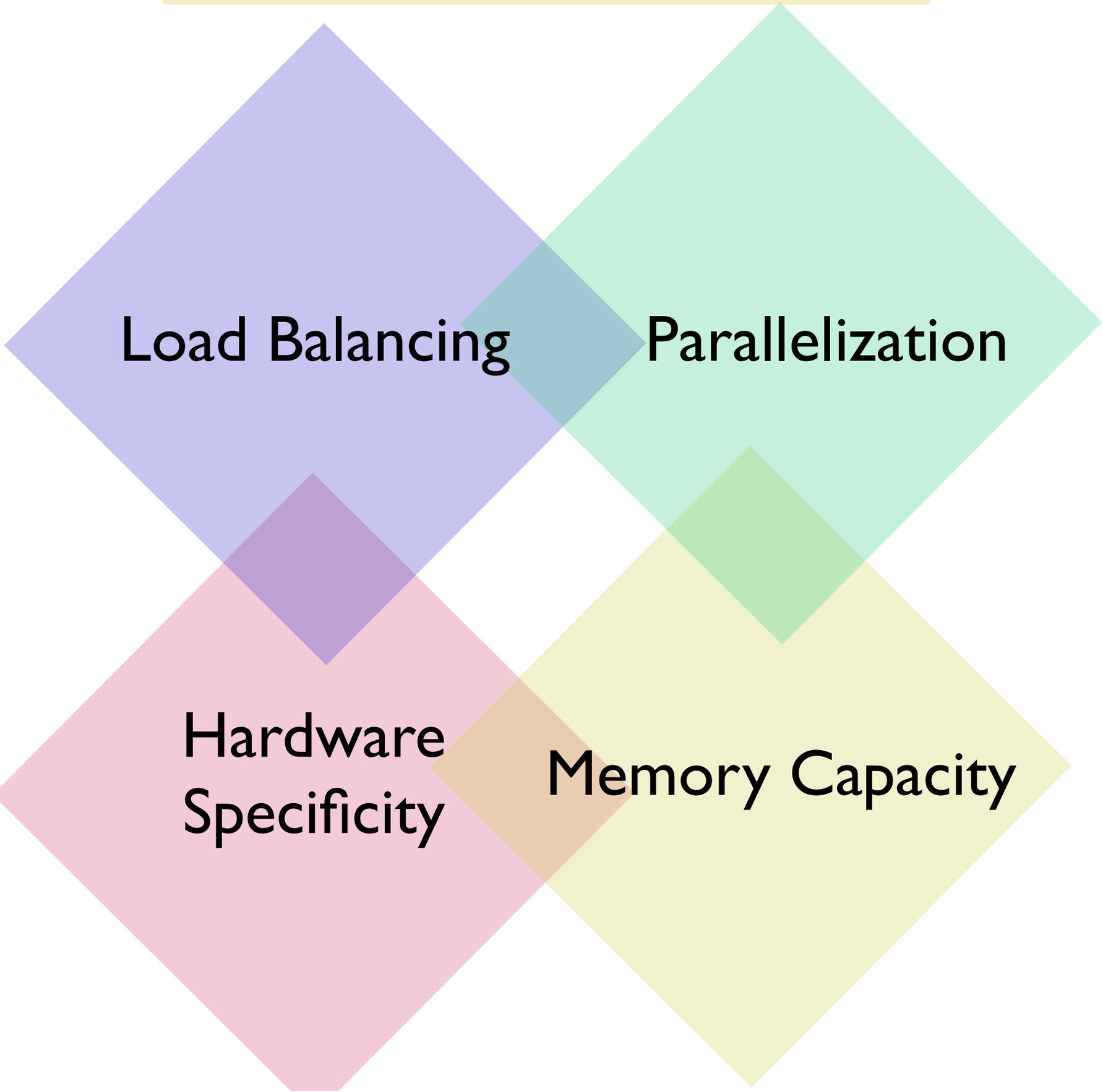
1.6 GHz Dual-core Atom

32-160 GB Flash SSD

Only 1 GB DRAM!

"Each decimal order of magnitude increase in parallelism requires a major redesign and rewrite of parallel code" - Kathy Yelick

The FAWN Quad of Pain

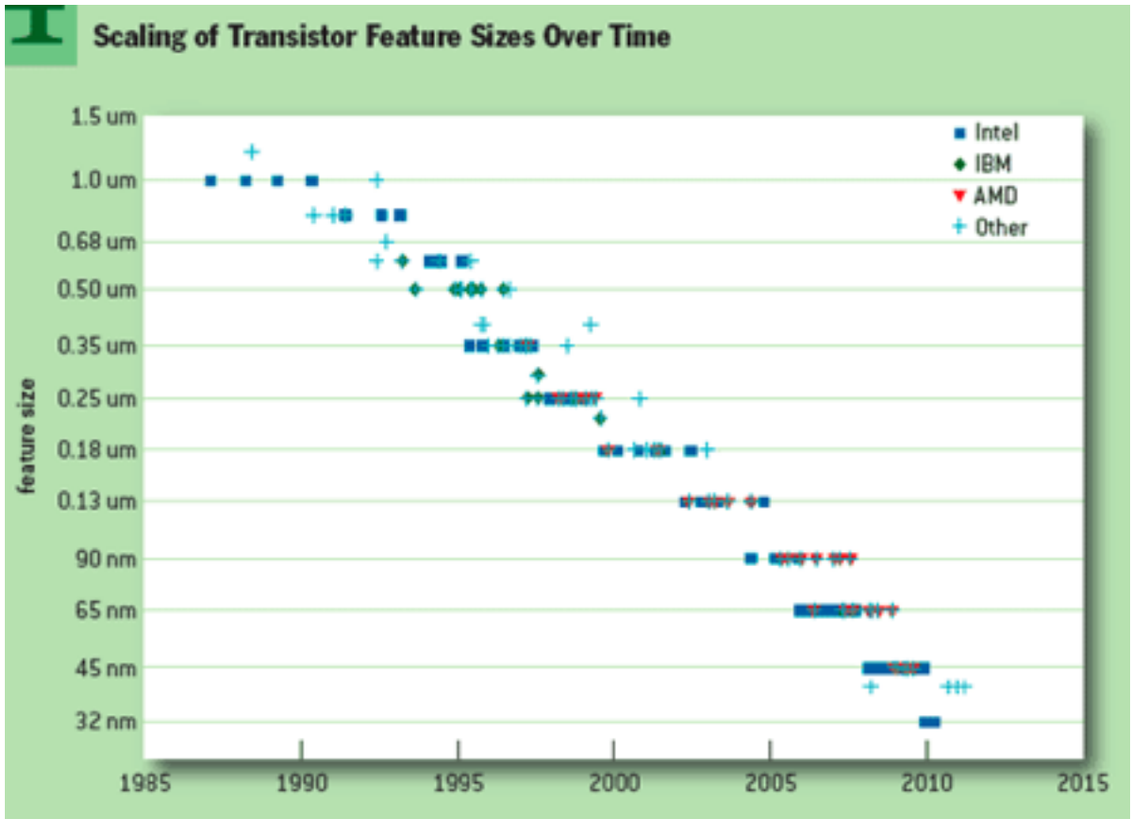


Bigger Clusters

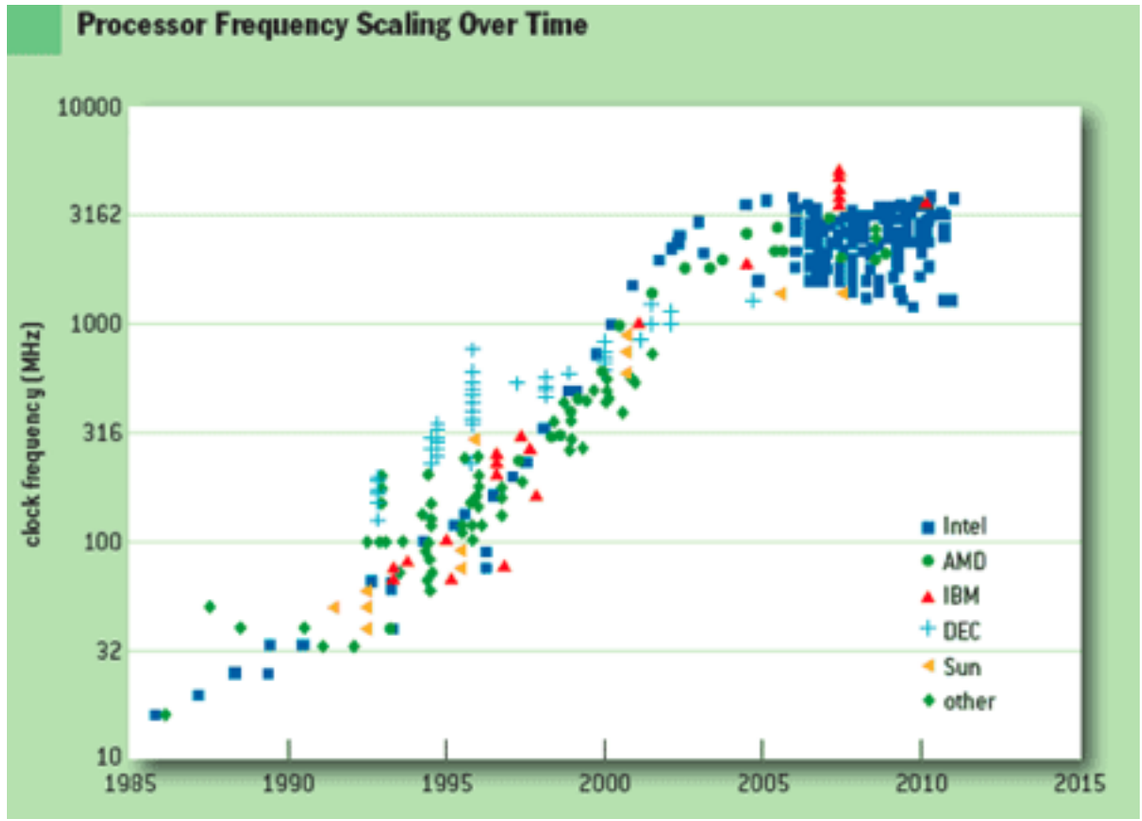
Wimpy Nodes

It's not just masochism

Moore



Dennard



(Figures from Danowitz, Kelley, Mao, Stevenson, and Horowitz: CPU DB)

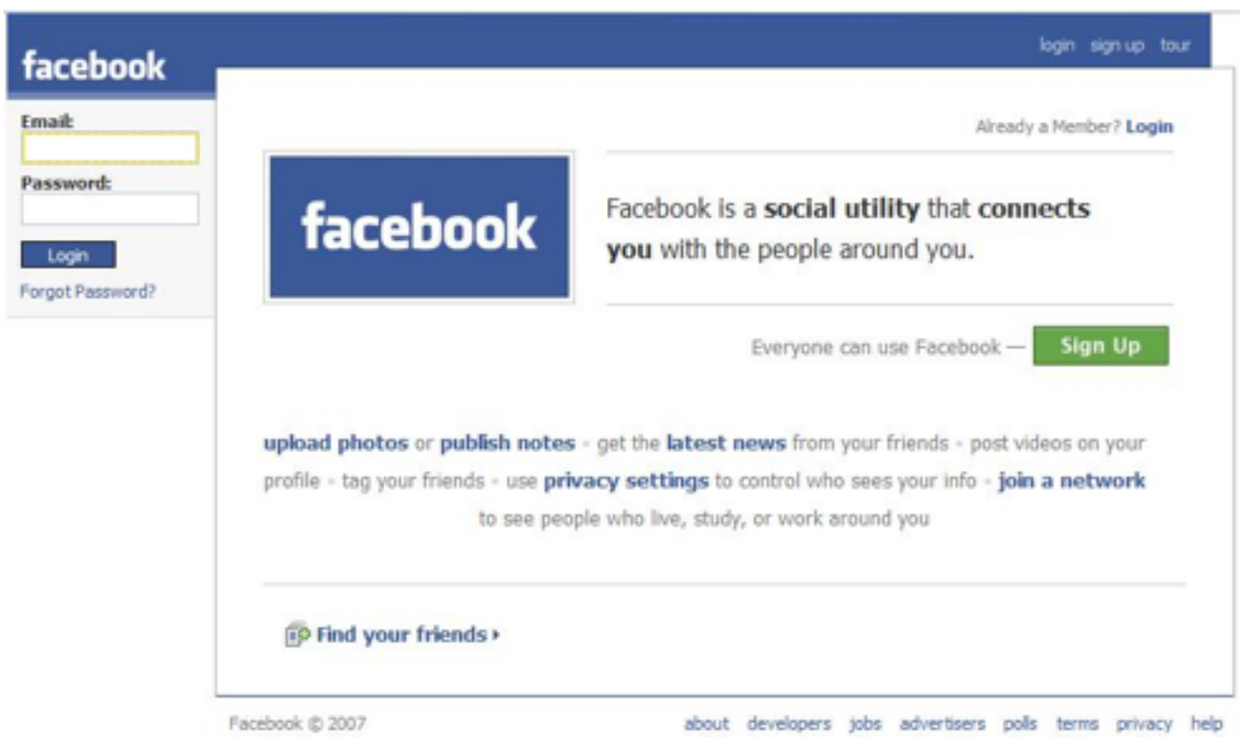
All systems will face this challenge over time

FAWN:

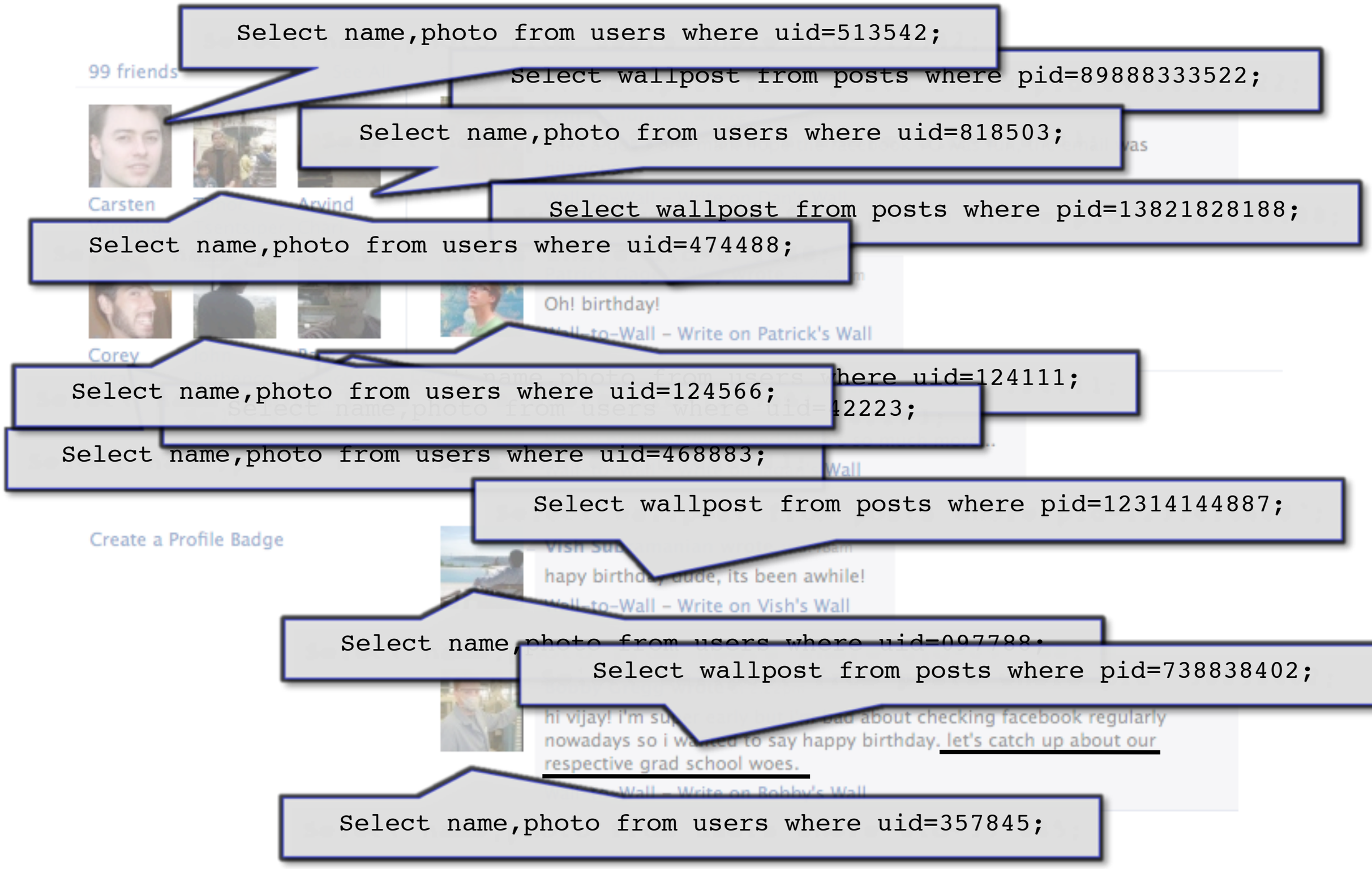
It started with a key-value store

Key-value storage systems

- Critical infrastructure service
- Performance-conscious
- Random-access, read-mostly, hard to cache



Small record, random access



FAWN-DS FAWN-KV SILT Small Cache Cuckoo

key-value backend store

Parallel fast

memory-efficient

hyper-optimized

optimized for flash

Mimics a cache

and large flash

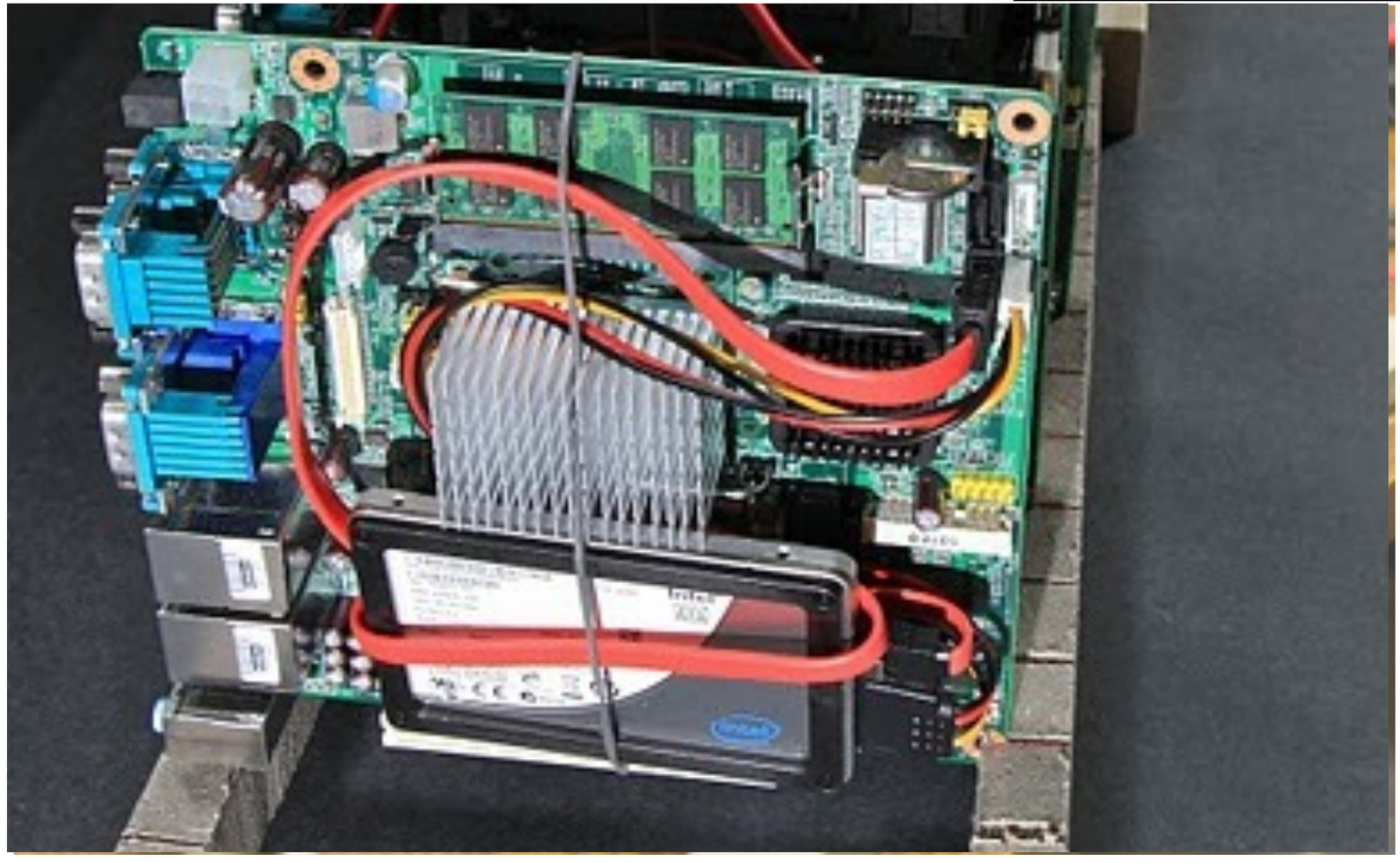
cuckoo hashing

Cuckoo Cache

Fawn-DS SILT

Fawn-DS SILT

Fawn-DS SILT

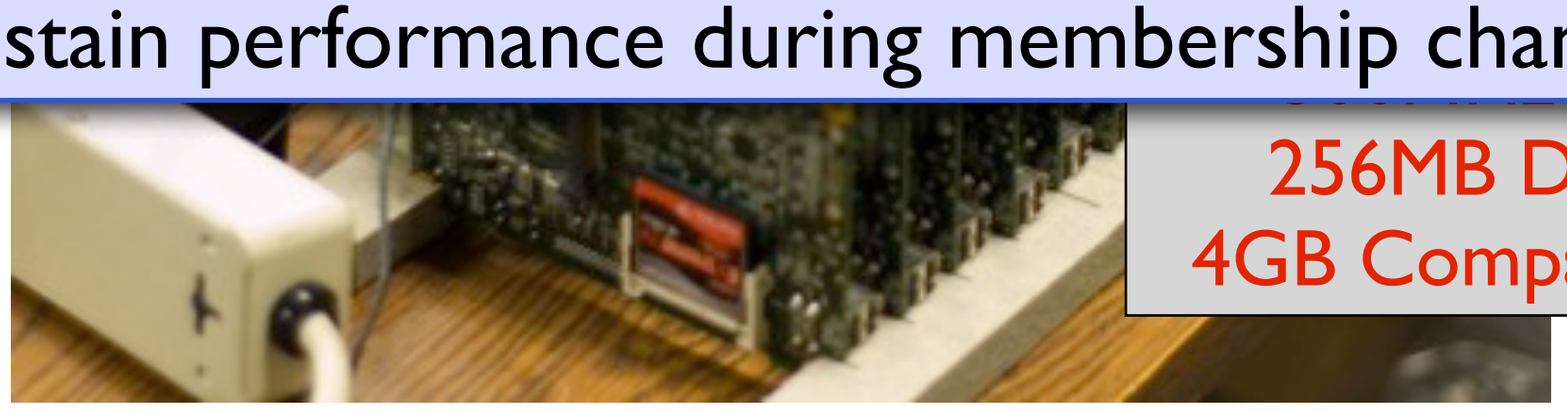


FAWN-DS and -KV:
Key-value Storage System

Goal: improve **Queries/Joule**

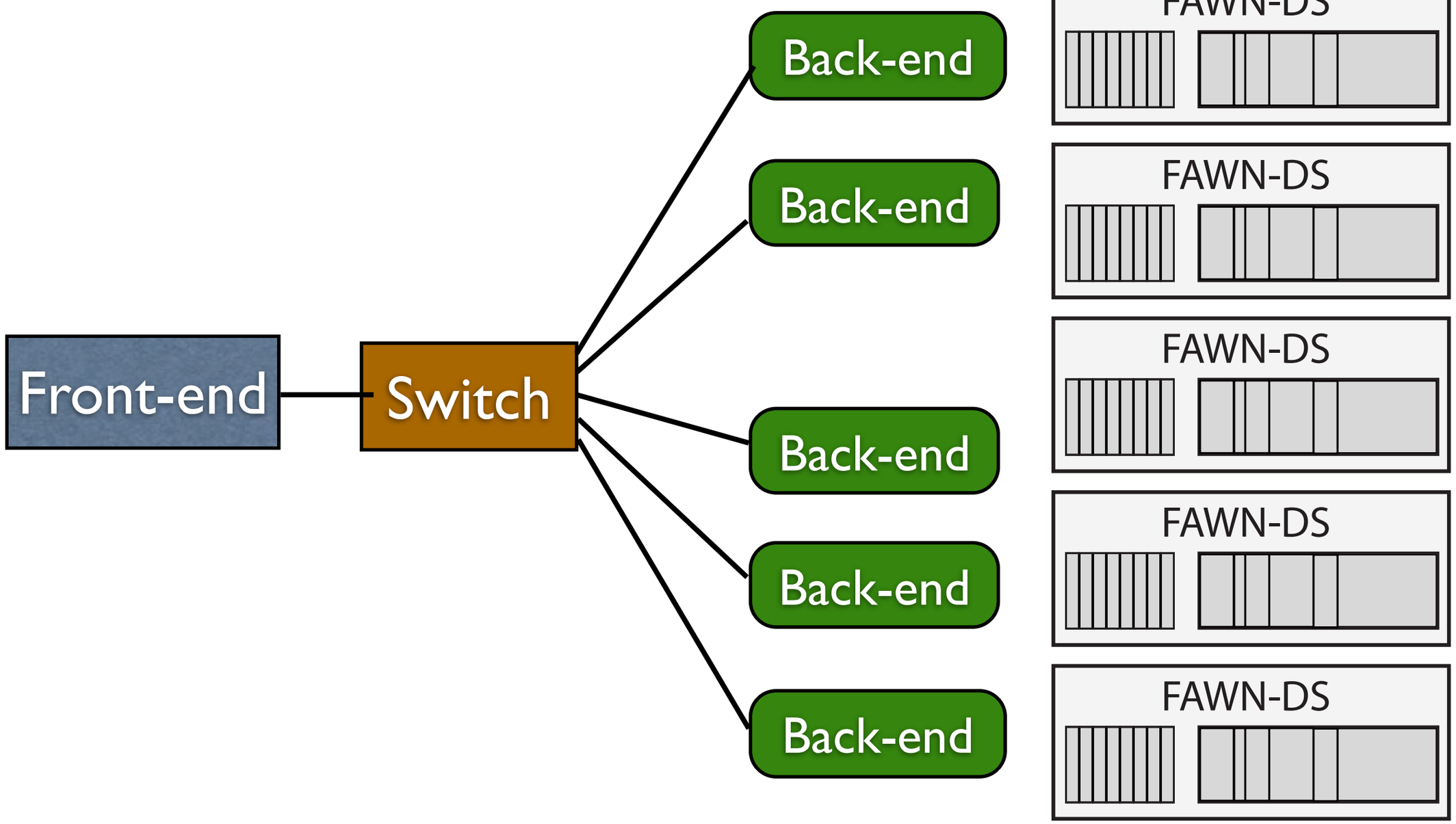
Unique Challenges:

- Wimpy CPUs, limited DRAM
- Flash poor at small random writes
- Sustain performance during membership changes

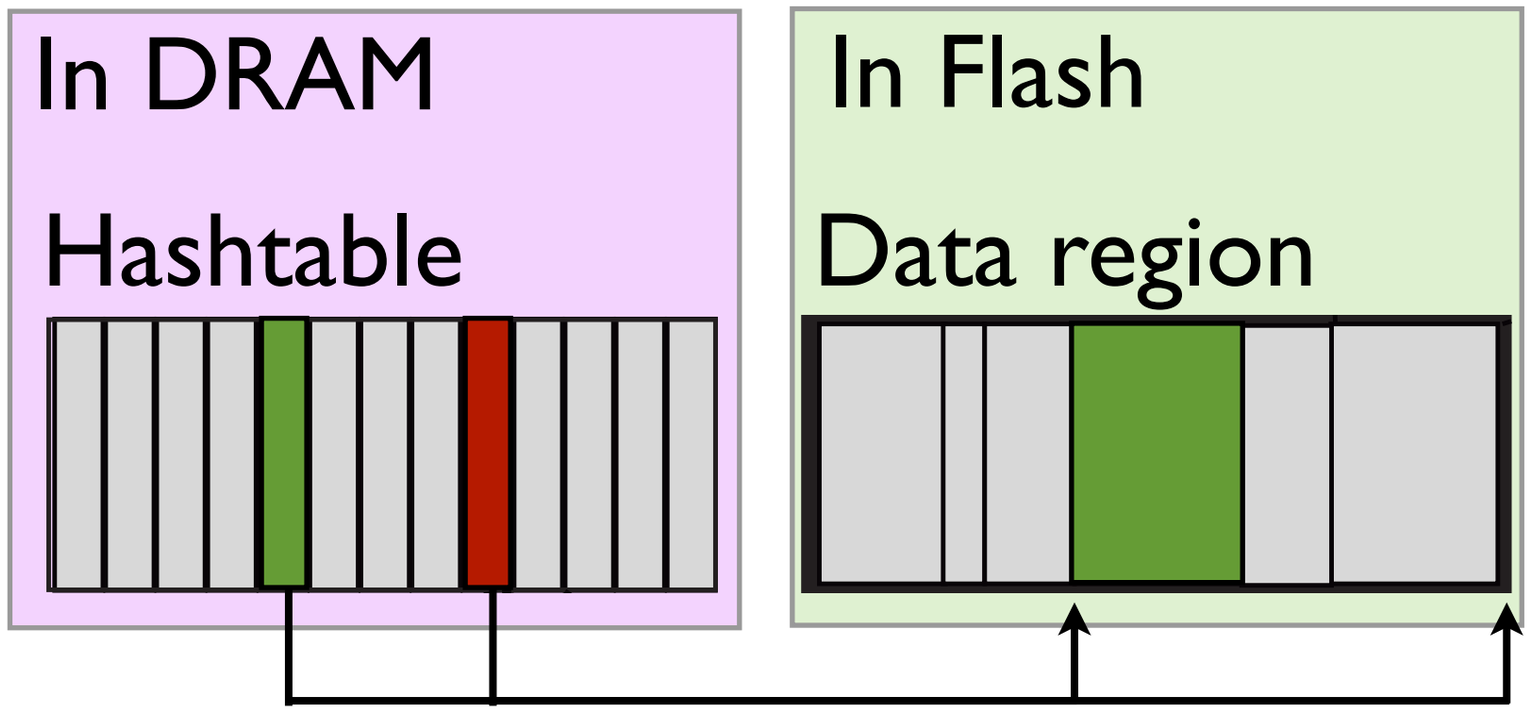


256MB DRAM
4GB CompactFlash

FAWN-KV Architecture



Avoiding random writes



Put K_1,V

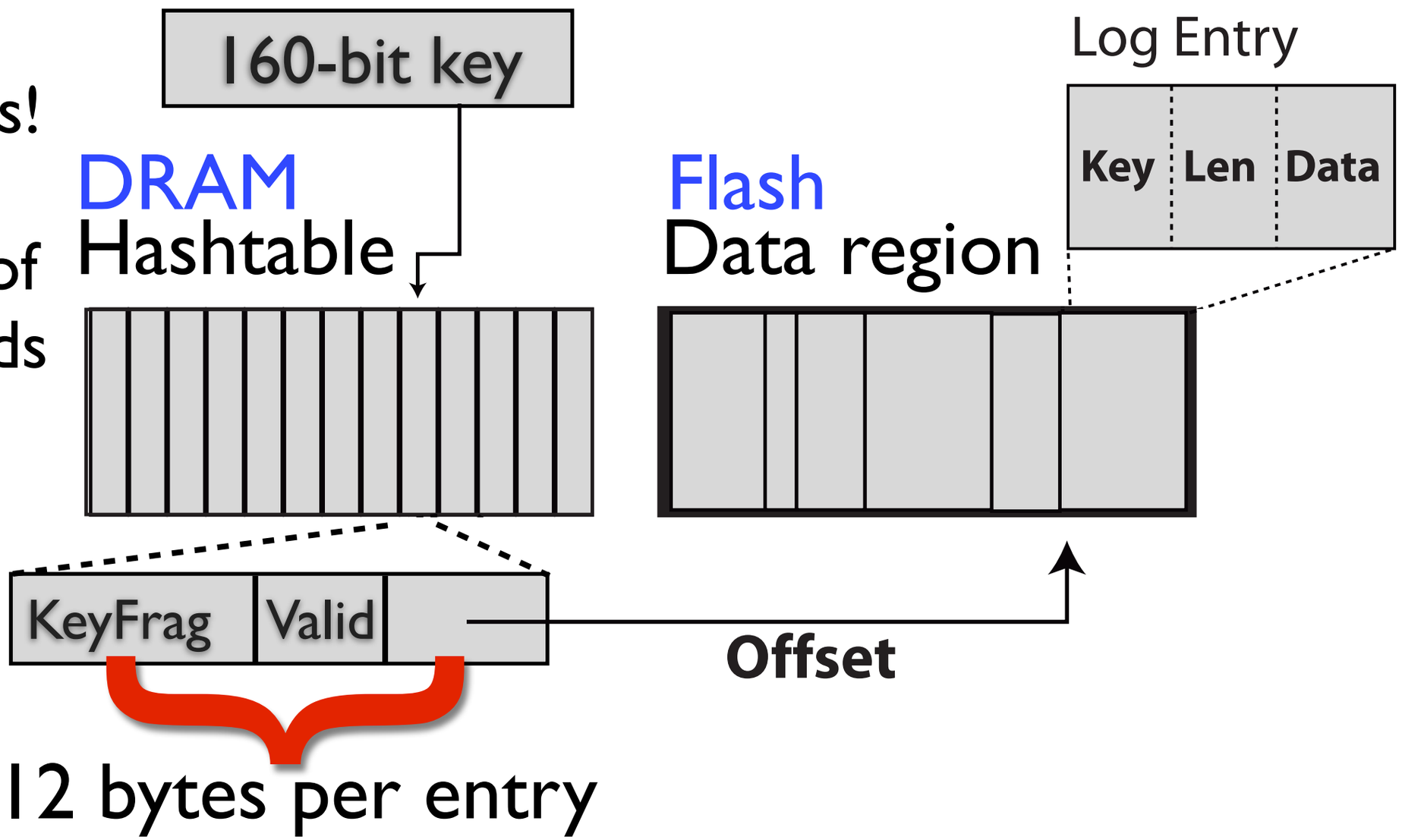
Get K_2

All writes to Flash are sequential

With limited DRAM

KeyFrag != Key
Potential collisions!

Low probability of multiple Flash reads



Partial-key hashing

Evaluation Takeaways

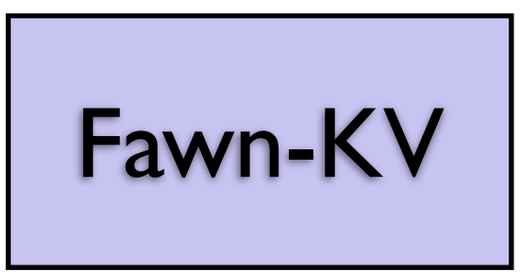
- 2008: FAWN-based system 6x more efficient than traditional systems
  - Partial-key hashing enabled memory-efficient DRAM index for flash-resident data
- Can create high-peformance, predictable storage service for small key-value pairs

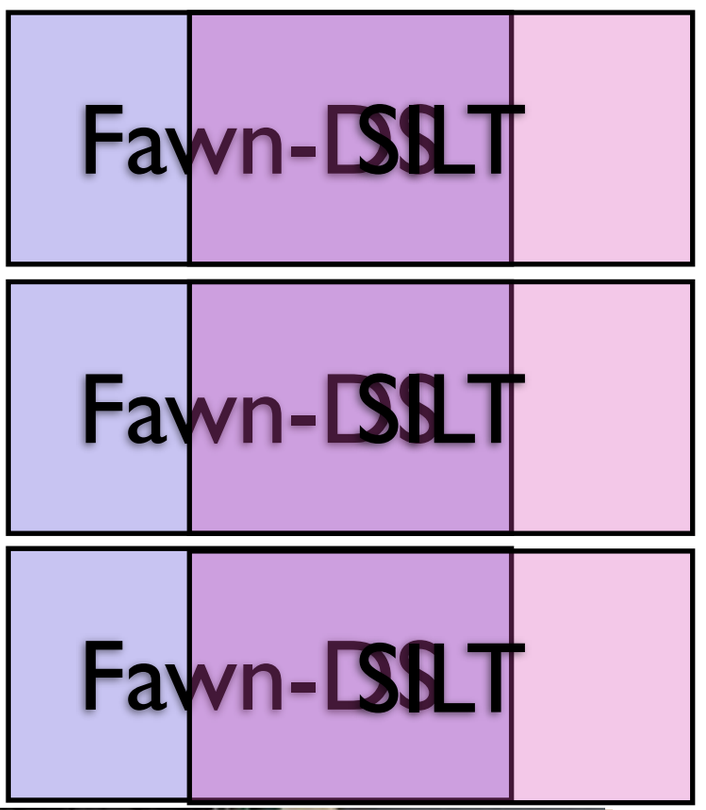
And then we moved to Atom + SSD

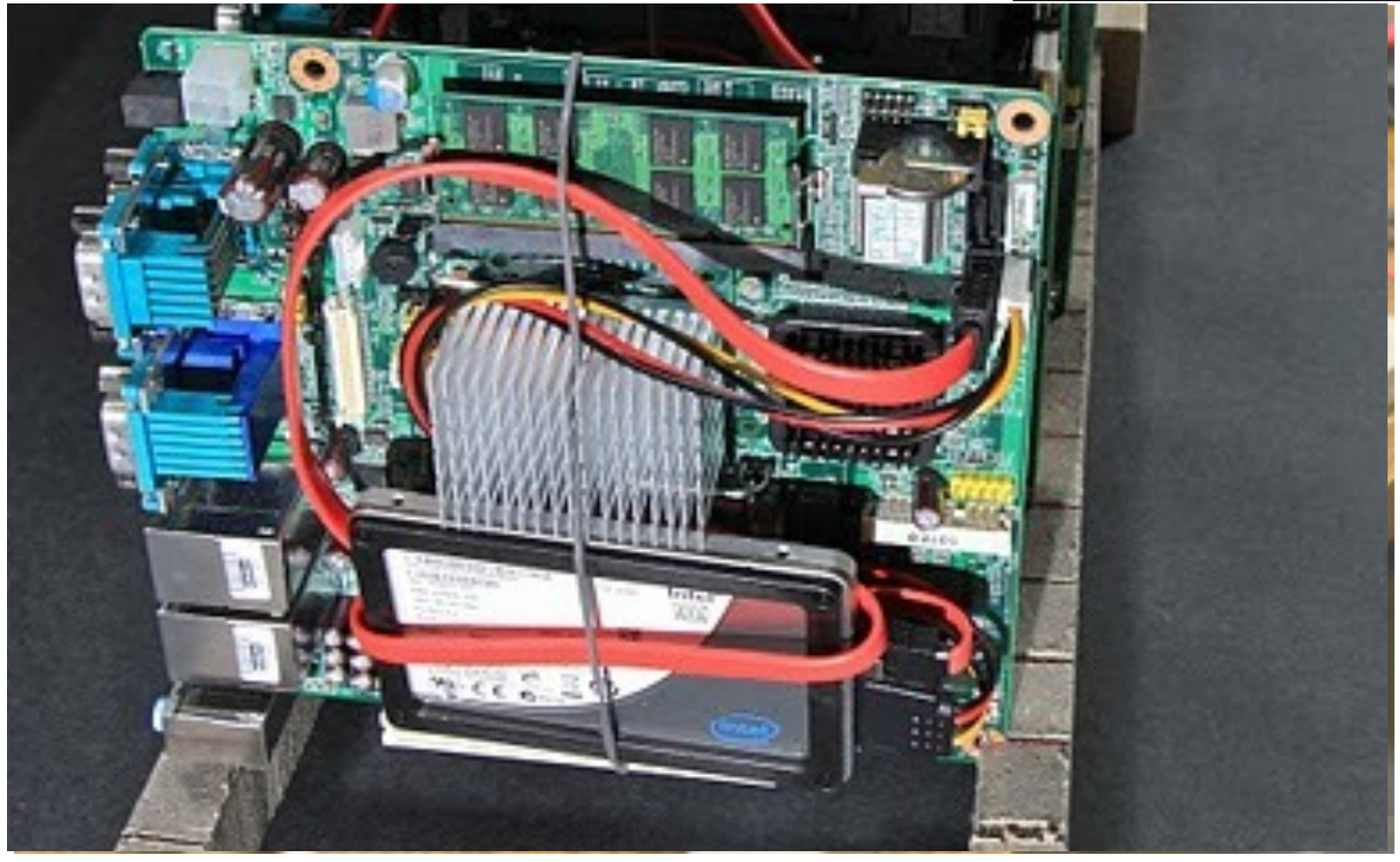
Geode 500Mhz	256MB	4GB CF Card ~2k IOPS
6x	8x	30-60x
Atom 1.6 Ghz single-core	2GB	120GB SSD ~60k IOPS

FAWN-DS FAWN-KV SILT Small Cache Cuckoo

backend store
hyper-optimized
for low DRAM
and large flash







Flash Must be Used Carefully

Random reads / sec	48,000

→ Fast, but not <u>THAT</u> fast

$ / GB	1.83

→ Space is precious

Another long-standing problem:

random writes are slow and bad for flash life (wearout)

Three Metrics to Minimize

Memory overhead = Index size per entry

- Ideally 0 bytes/entry (no memory overhead)

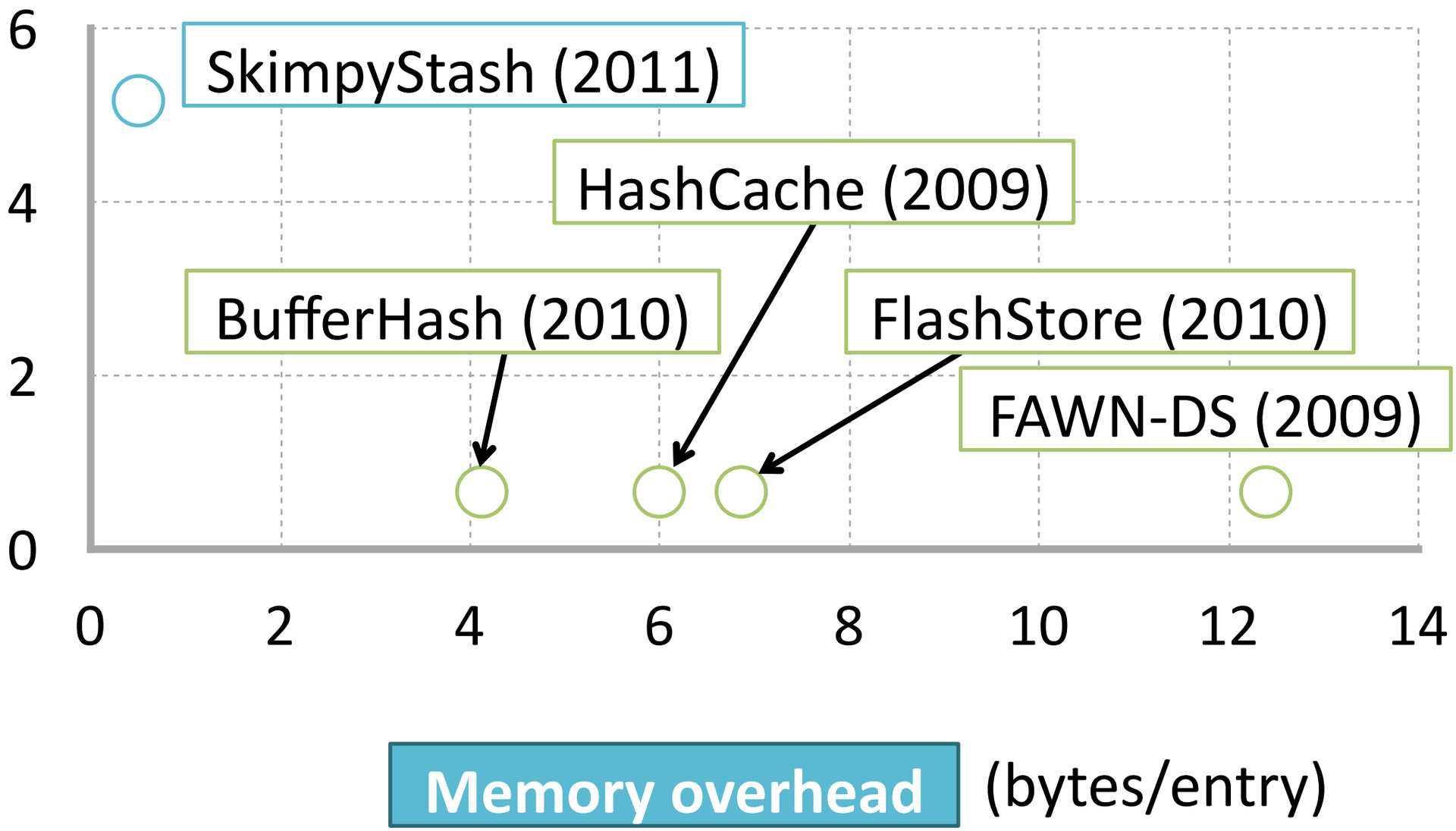
Read amplification = Flash reads per query

- Limits **query throughput**
- Ideally 1 (no wasted flash reads)

Write amplification = Flash writes per entry

- Limits **insert throughput**
- Also reduces **flash life expectancy**
  - Must be small enough for flash to last a few years

Read amplification



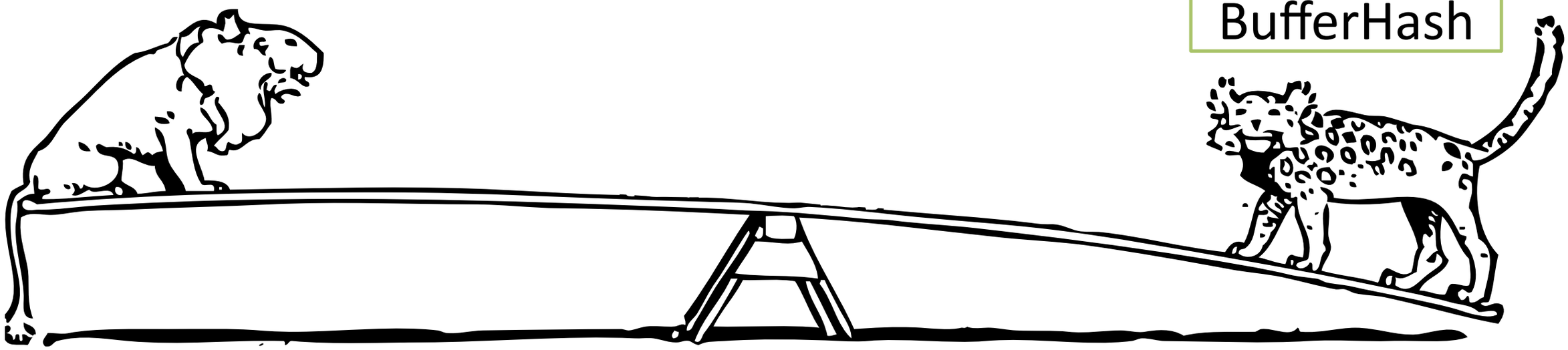
SkimpyStash

FAWN-DS

FlashStore

HashCache

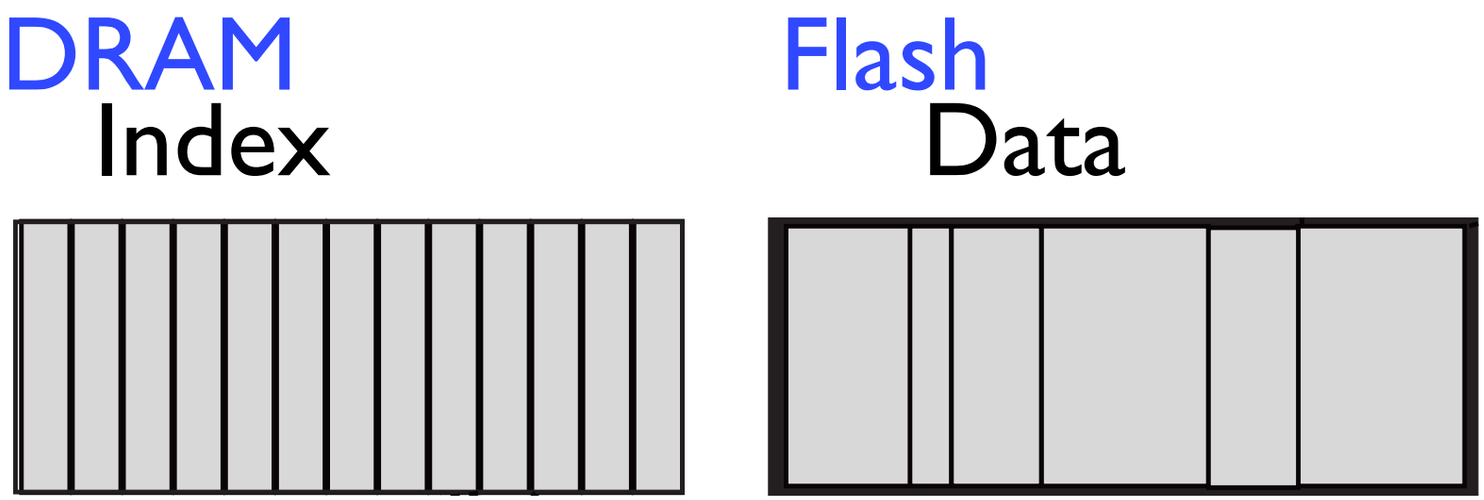
BufferHash



Memory efficiency

High performance

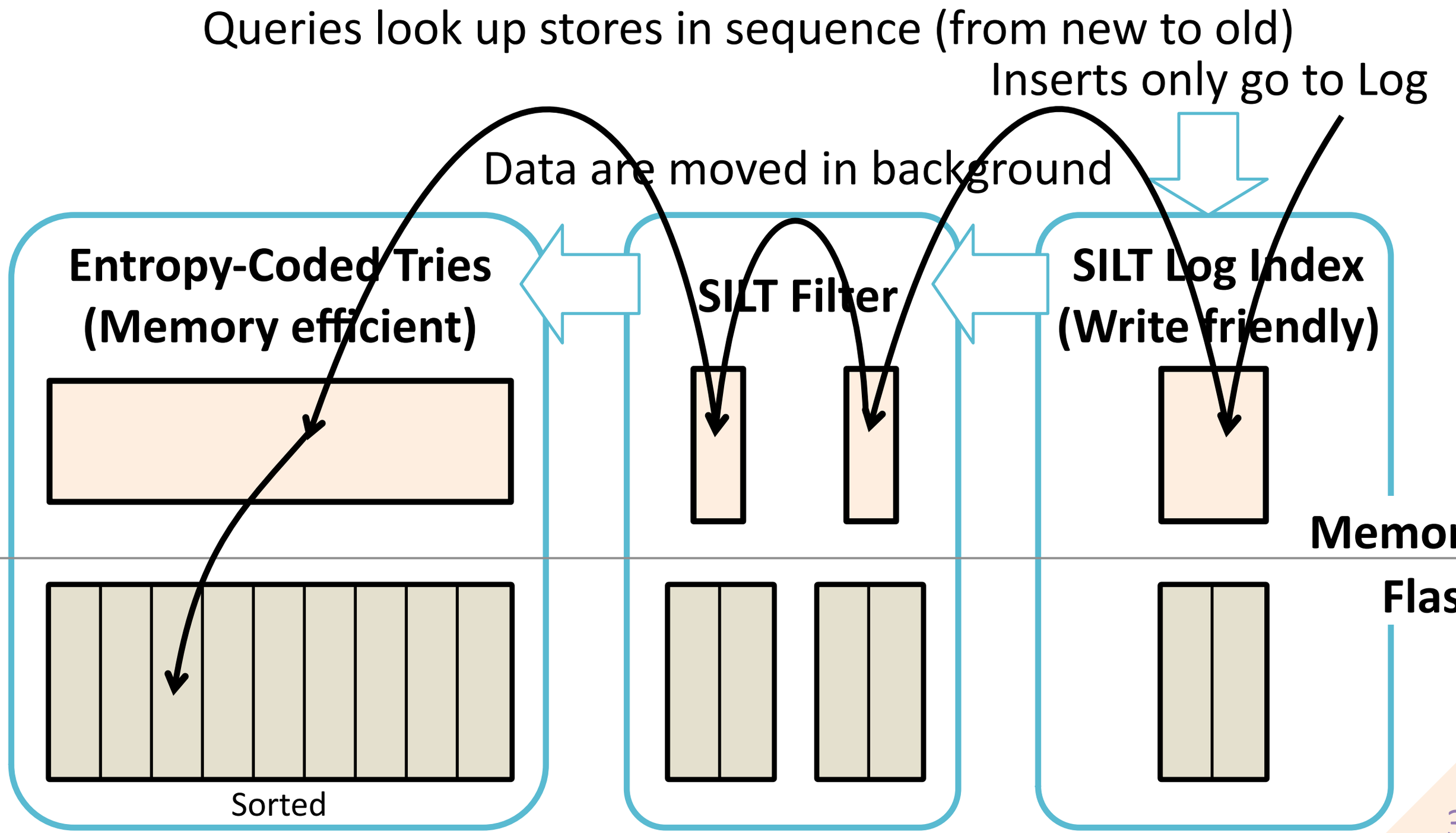
(static) “External Dictionary”

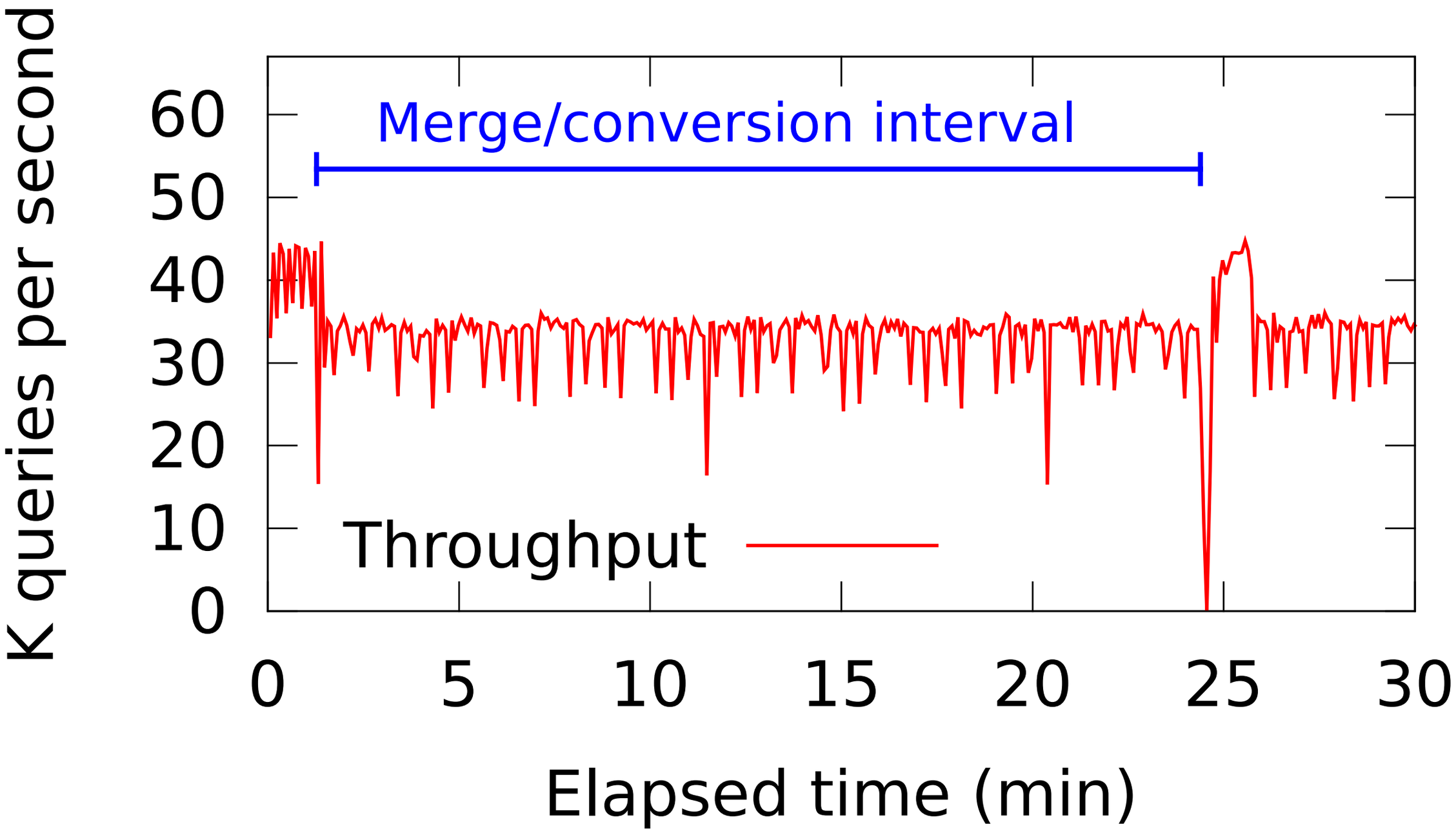


- Prior state of the art: “EPH”: ~3.8 bits/entry
- Ours: Entropy-coded tries, ~2.5 bits/entry

- Important considerations:
  - Construction speed; query speed
  - Aw, it’s read-only...

Solution: (1) Three Stores with (2) New Index Data Structures

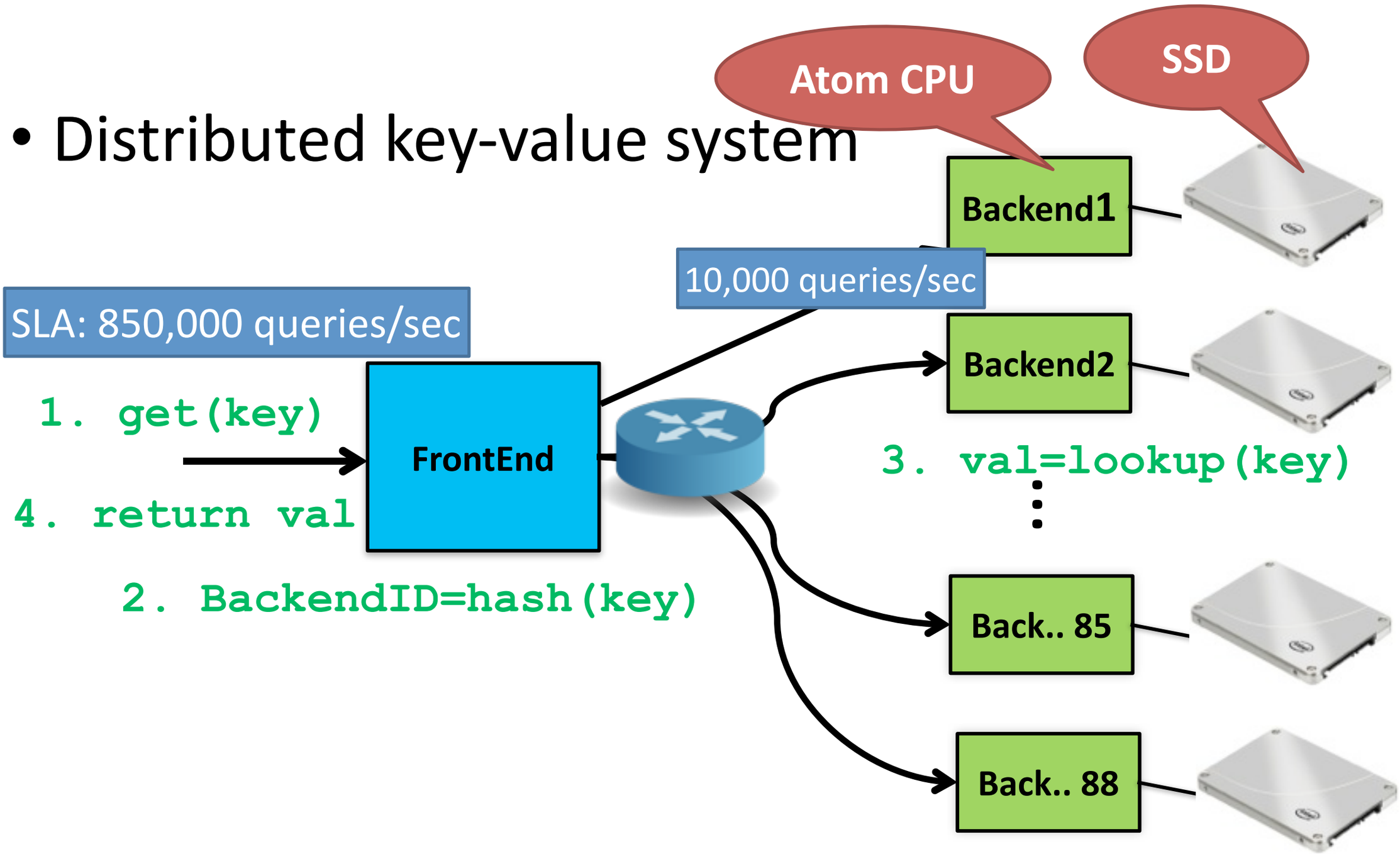




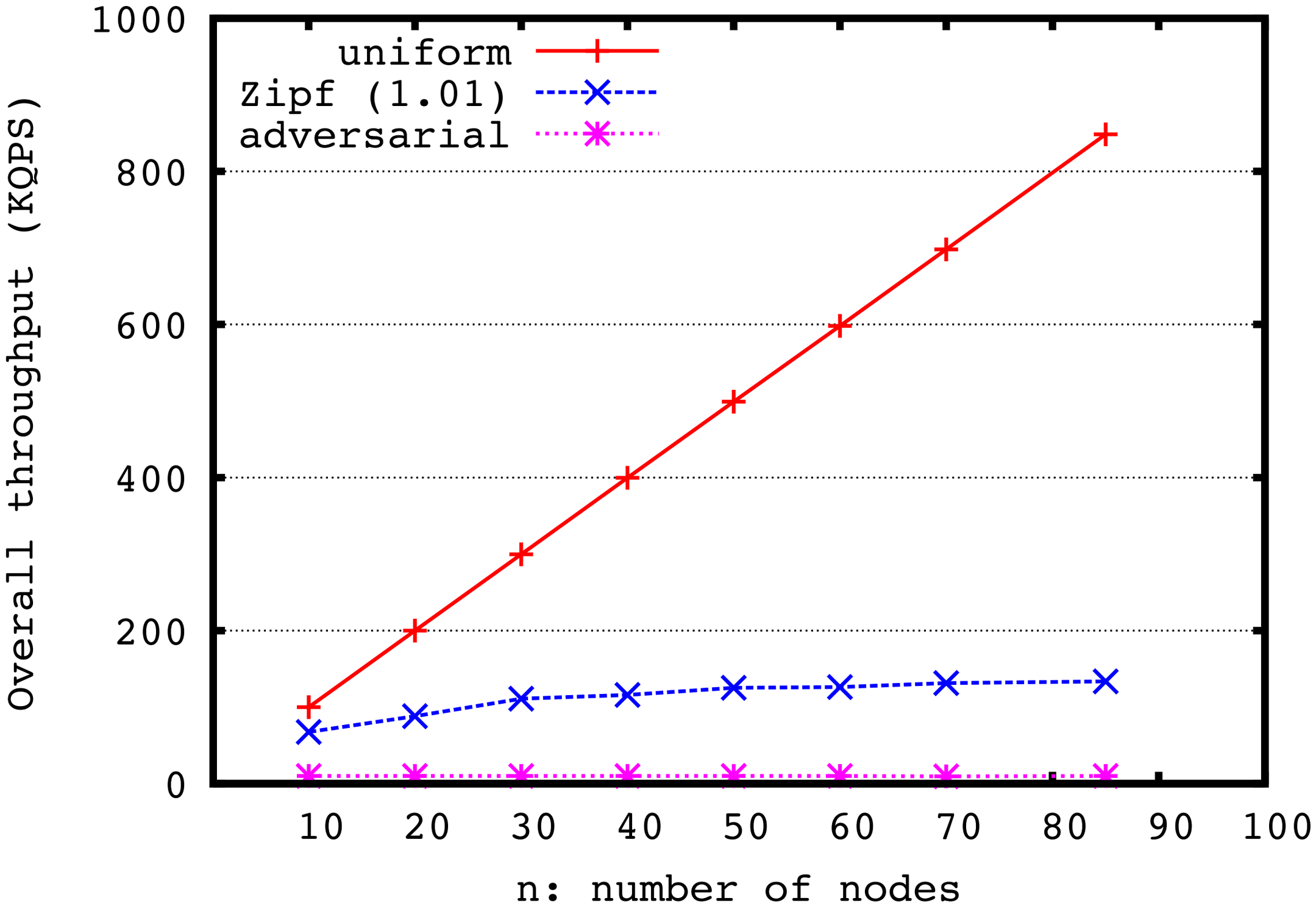
Caveat: Not on wimpies. Still working on reducing CPU cost! :-)

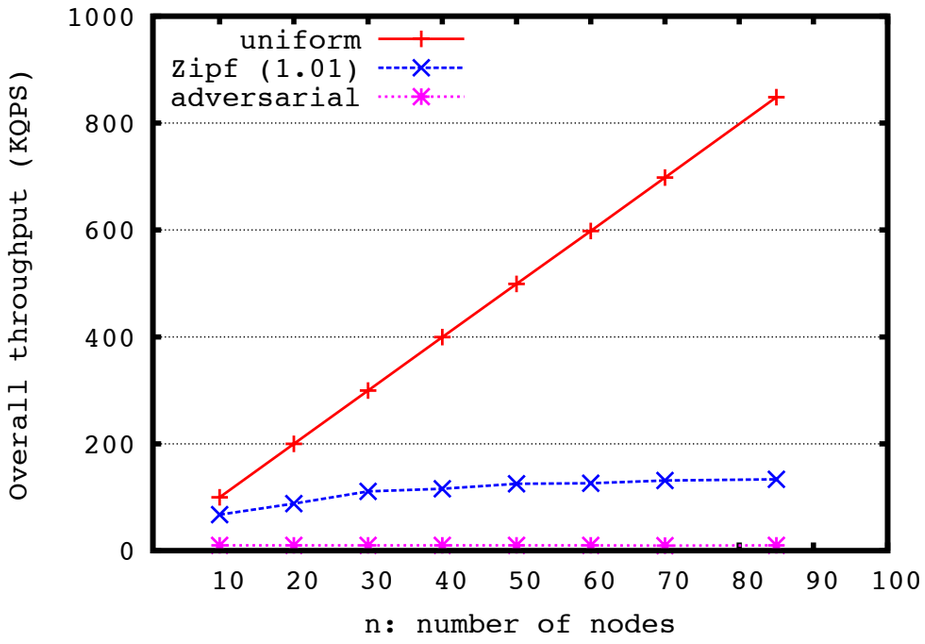
And now... Load imbalance

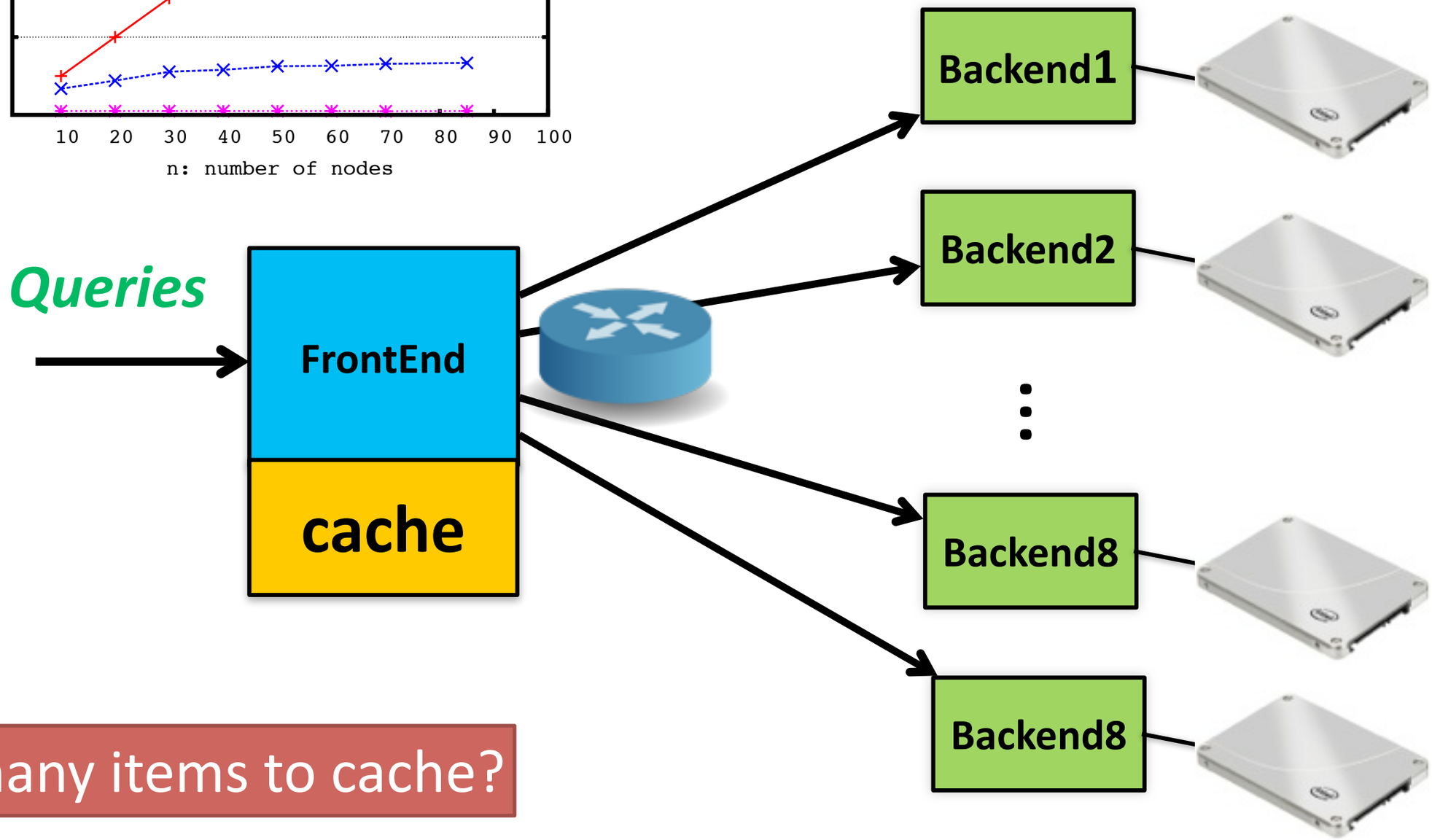
- Distributed key-value system



Measured tput on FAWN testbed



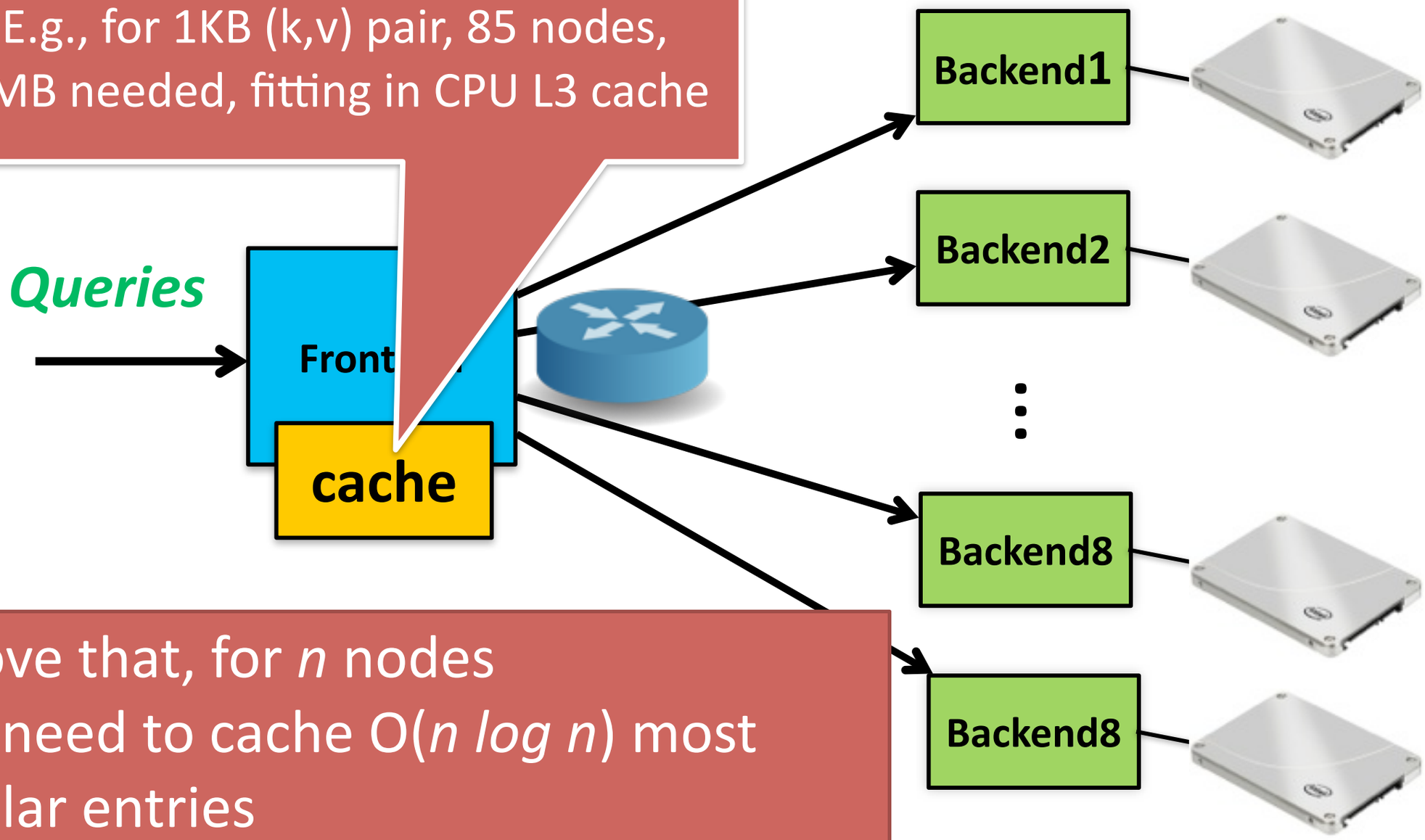




How many items to cache?

small/fast cache is enough!

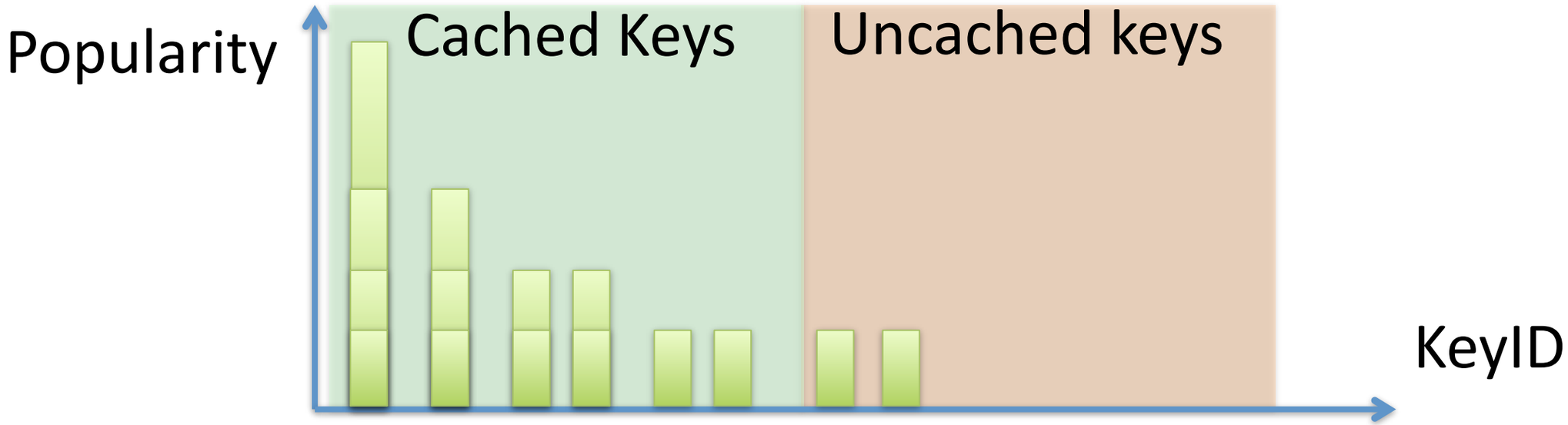
E.g., for 1KB (k,v) pair, 85 nodes, 3MB needed, fitting in CPU L3 cache



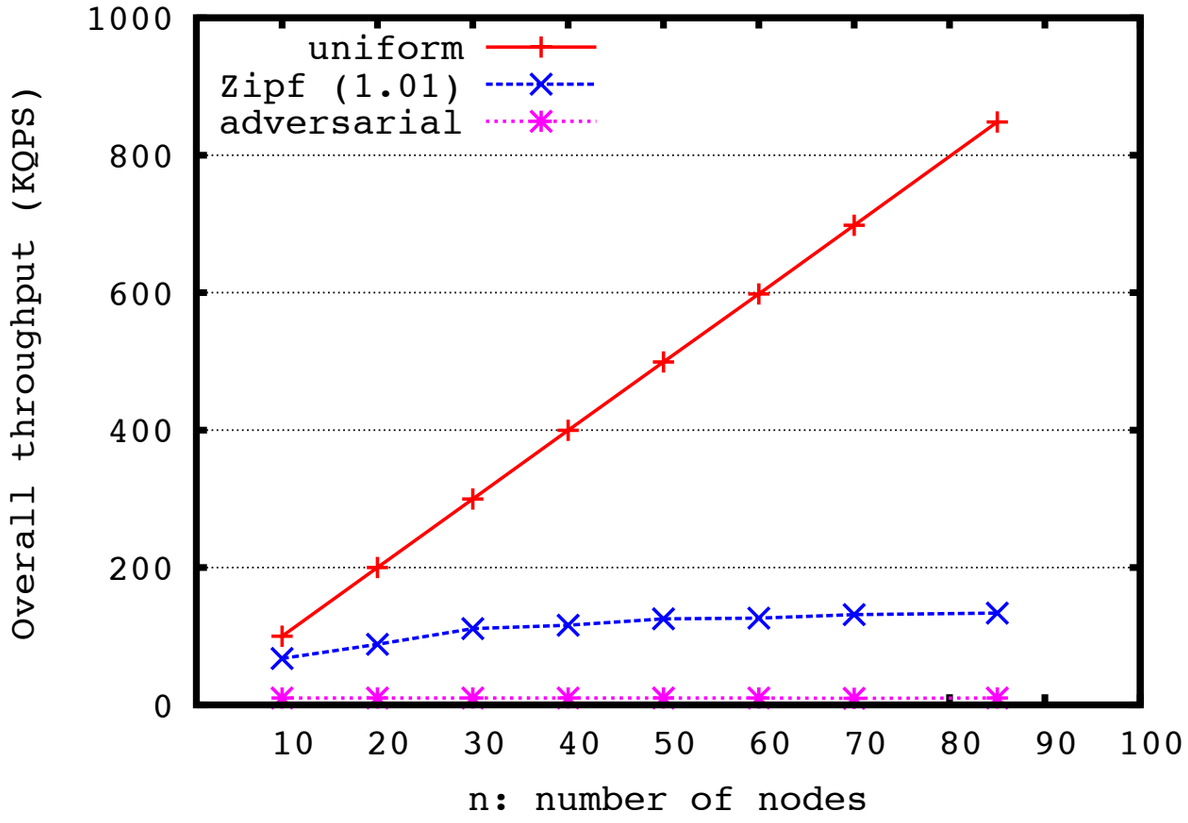
We prove that, for *n* nodes

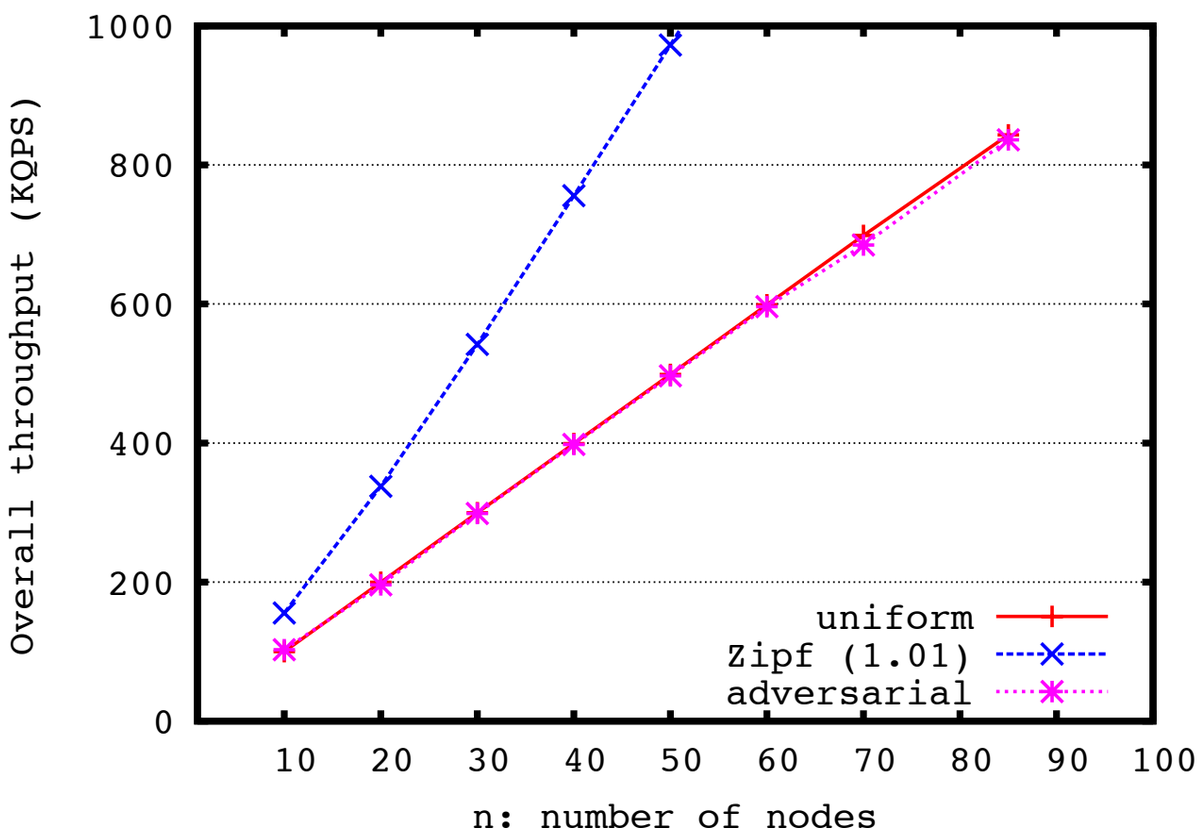
- Only need to cache O(*n log n*) most popular entries
- worst case perf. = *(1 - ε)* * n * single node capacity

Cache forces near-uniform dist.



Worst case? Now best case

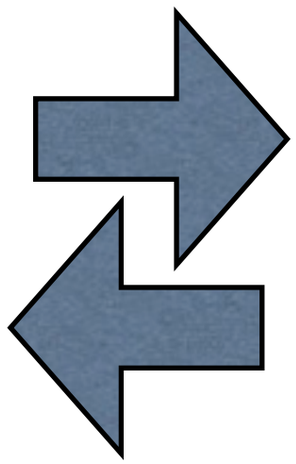




Thus…

"Wimpy" servers [FAWN, SOSP 2009]

"Brawny" server



Insanely Fast Cache

SILT

SILT

SILT

[SILT, SOSP 2011]

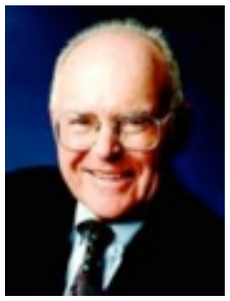
O(N log N) ["small cache" socc 2011]

Multi-reader parallel cuckoo hashing [under submission]

Entropy-coded tries [SILT + under submission]

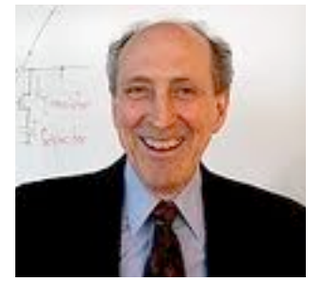
Partial-key cuckoo hashing

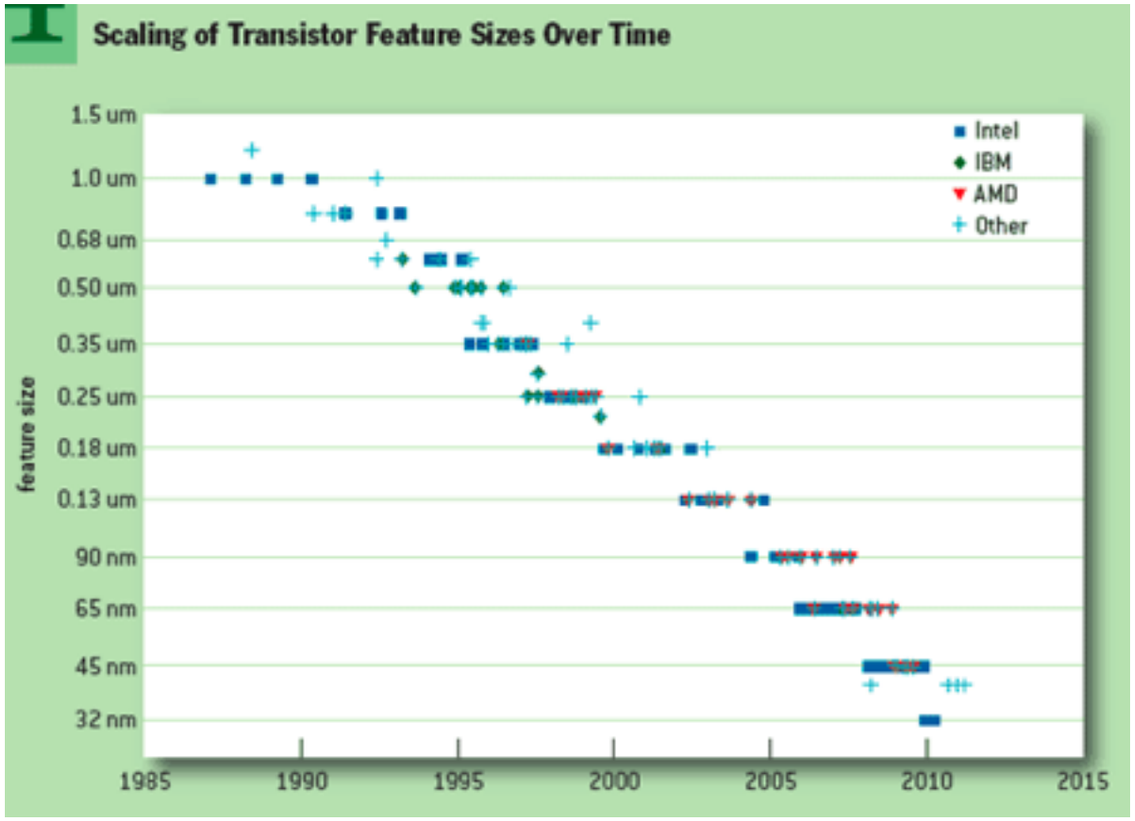
Cuckoo filter

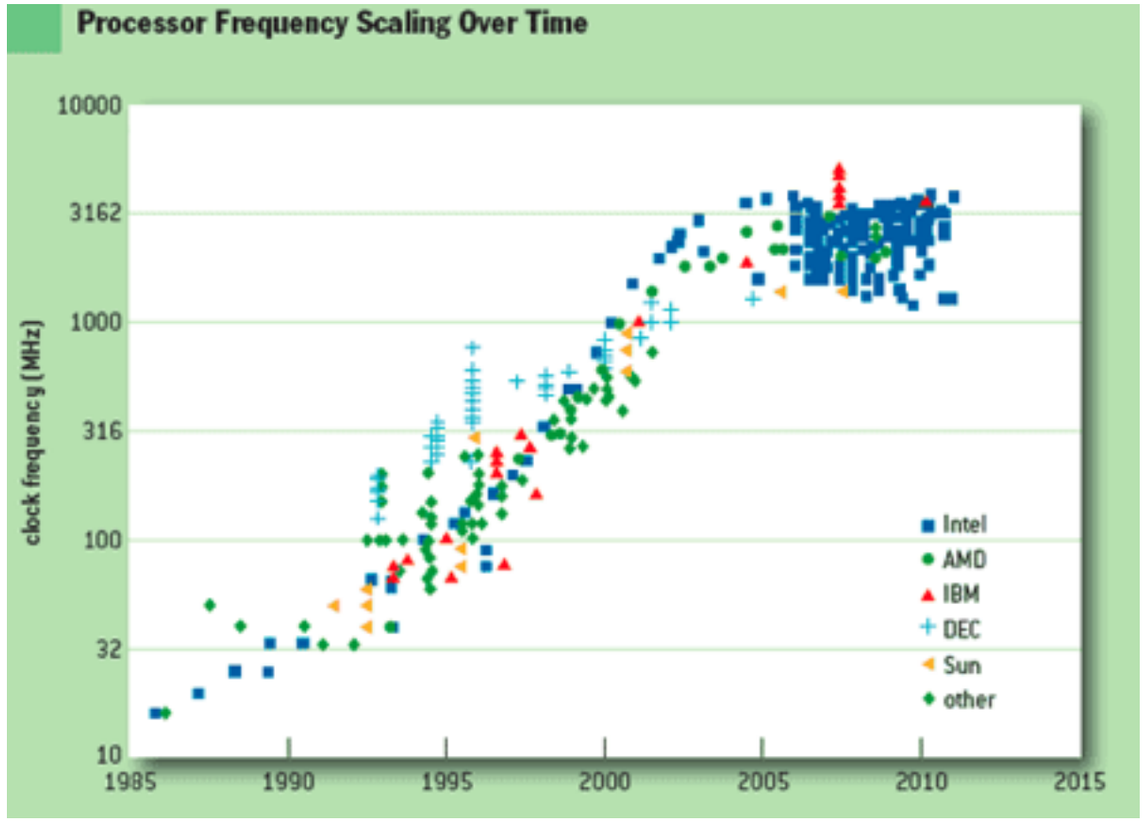


Moore

Dennard







highly parallel, lower-GHz, (memory-constrained?):

Architectures, algorithms, and programming