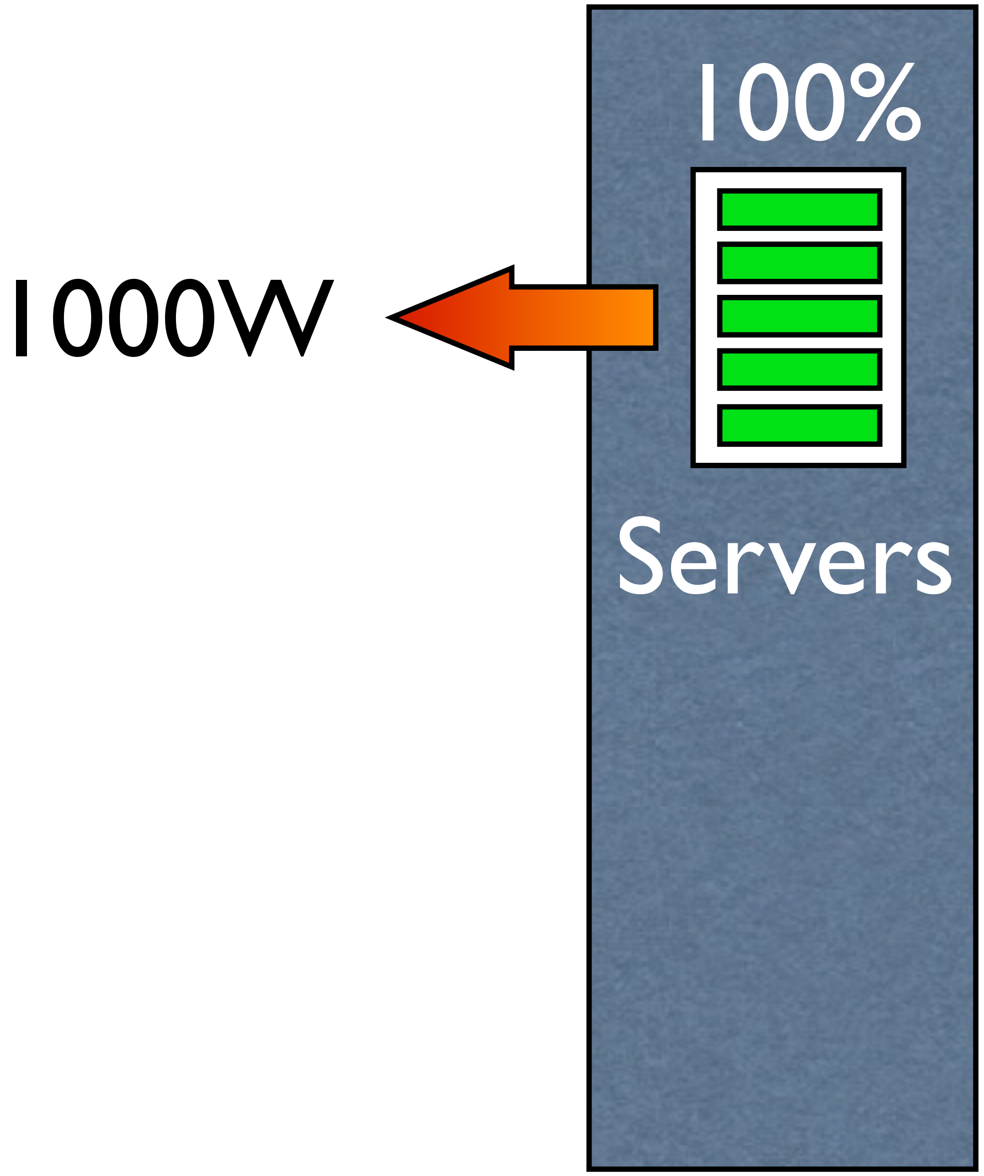
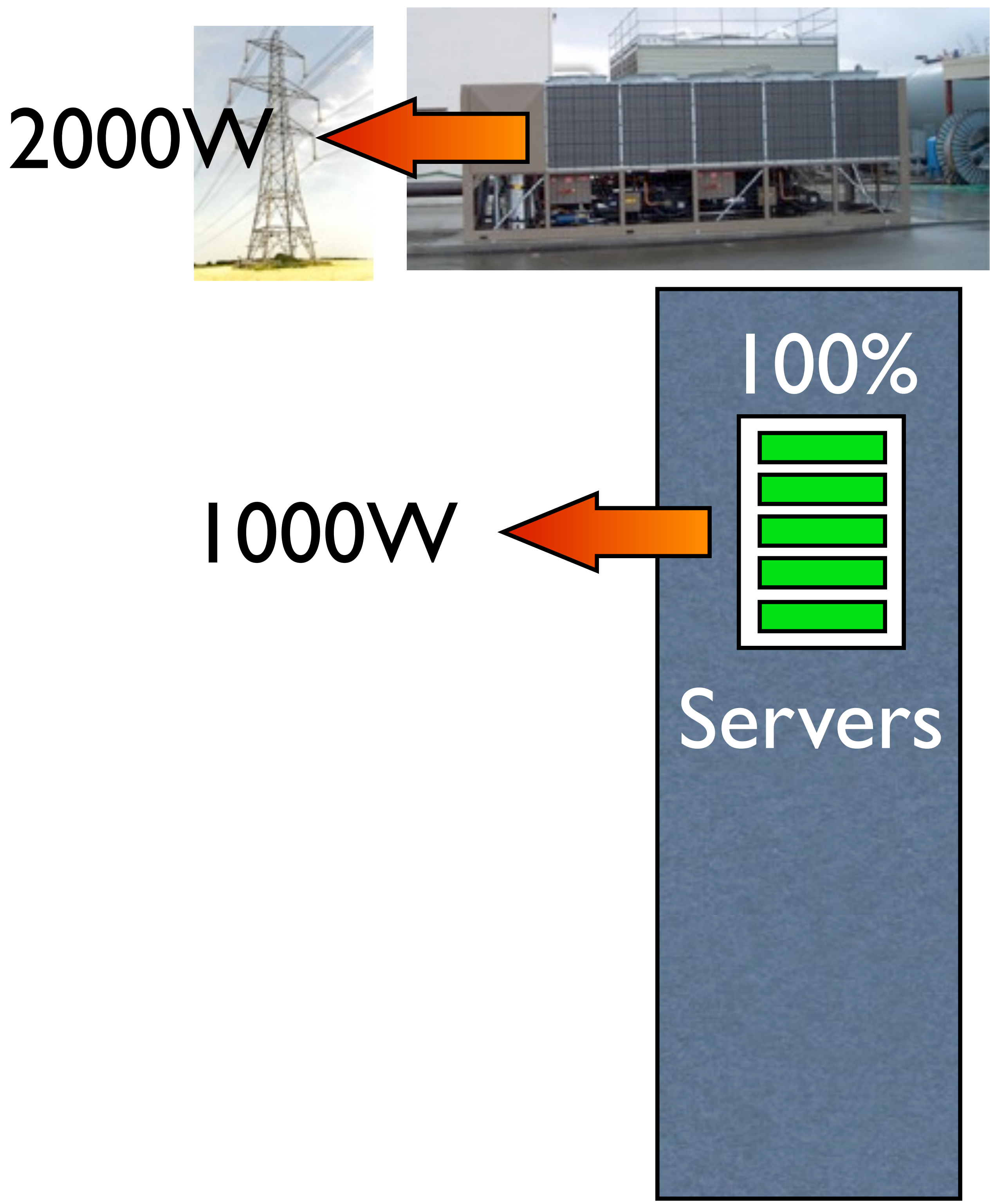
Learning to Scale Out By Scaling Down

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


1000W
100%
Servers

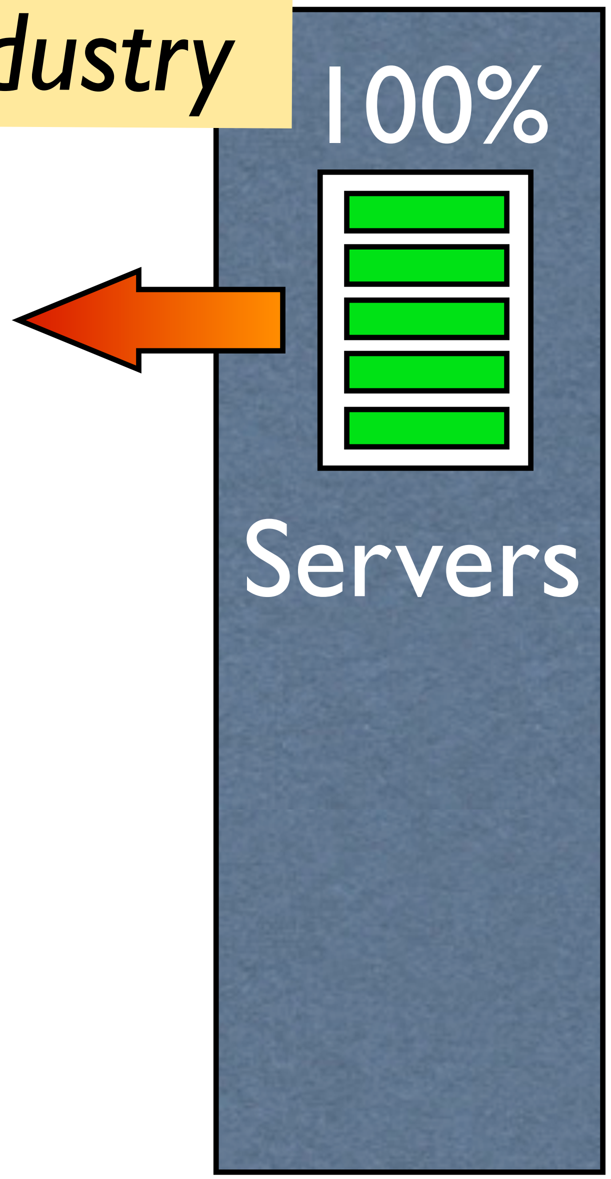

2000W
100%
1000W
Servers

Infrastructure:
PUE

2005: 2–3

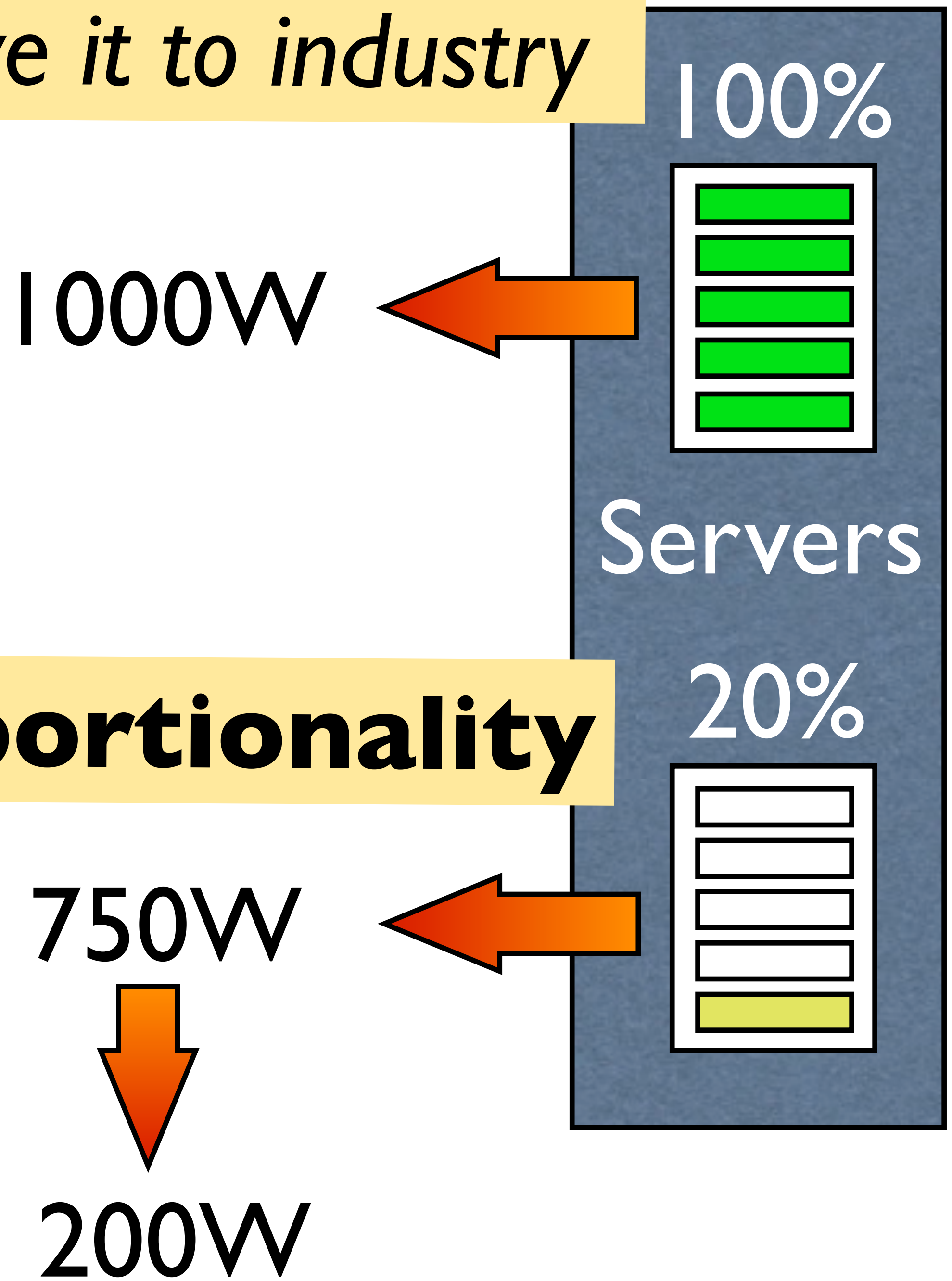
2012: ~1.1

Leave it to industry



Infrastructure:
PUE
2005: 2–3
2012: ~1.1
Leave it to industry

Proportionality

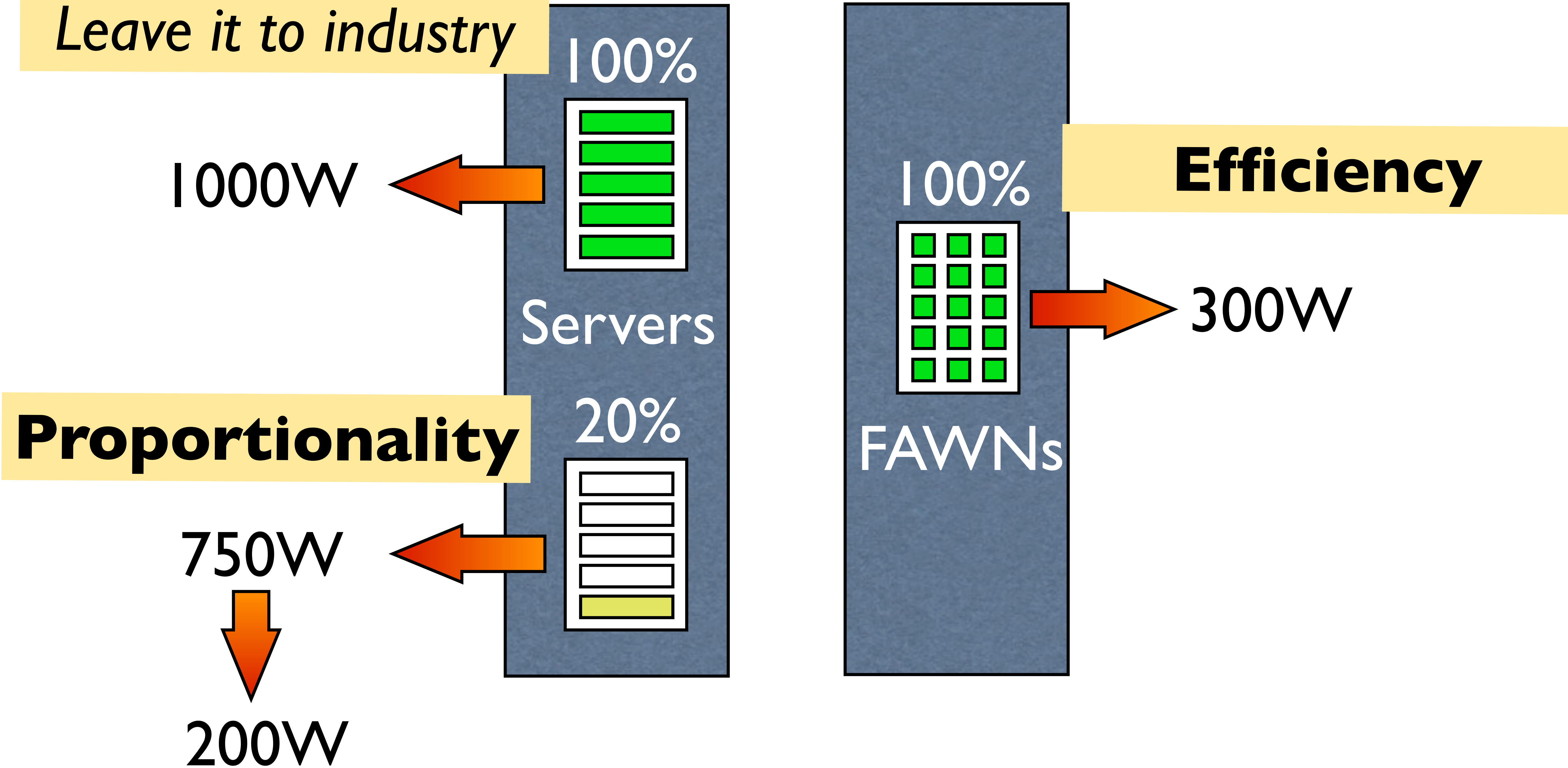


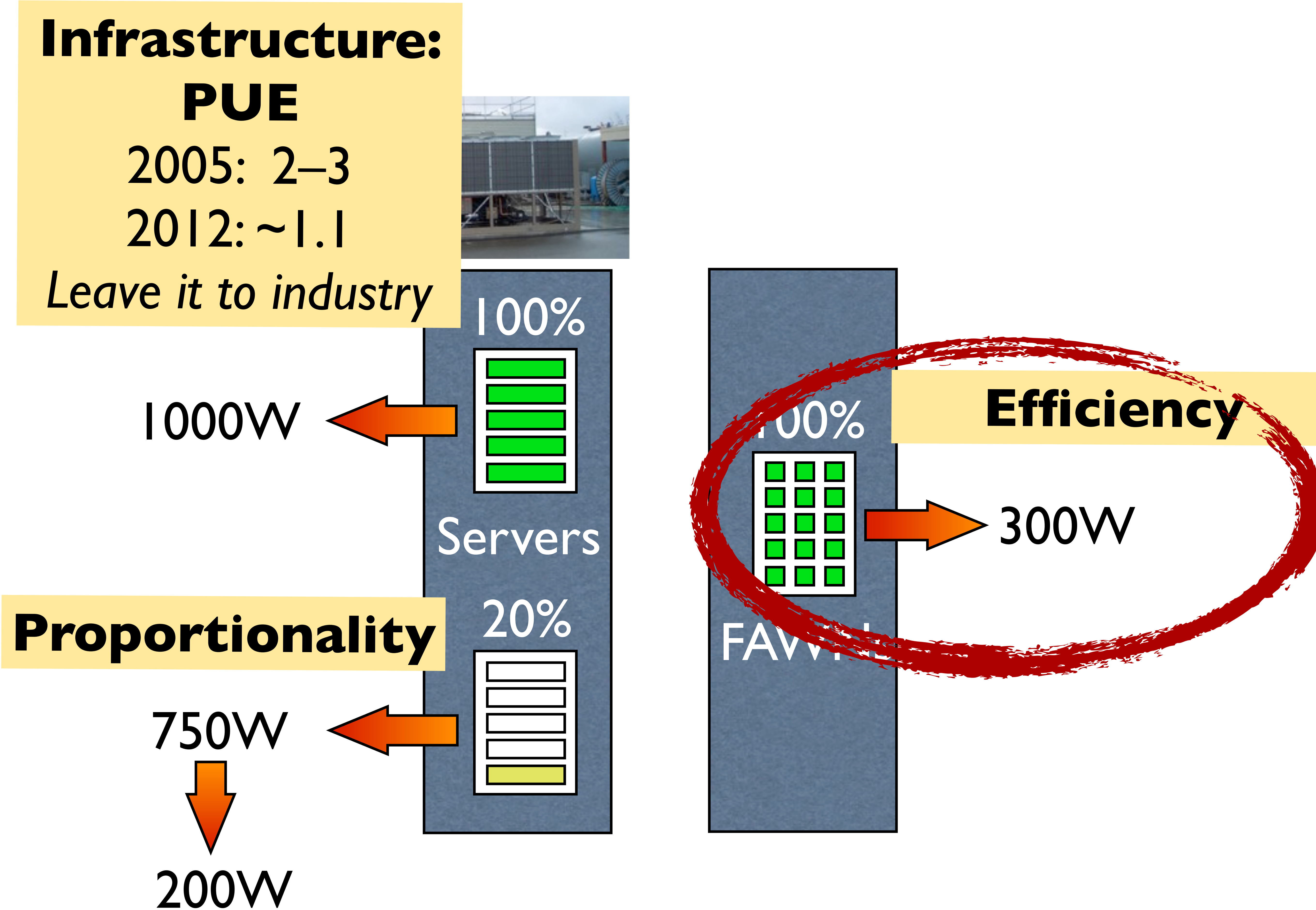
Infrastructure:
PUE

2005: 2–3

2012: ~1.1

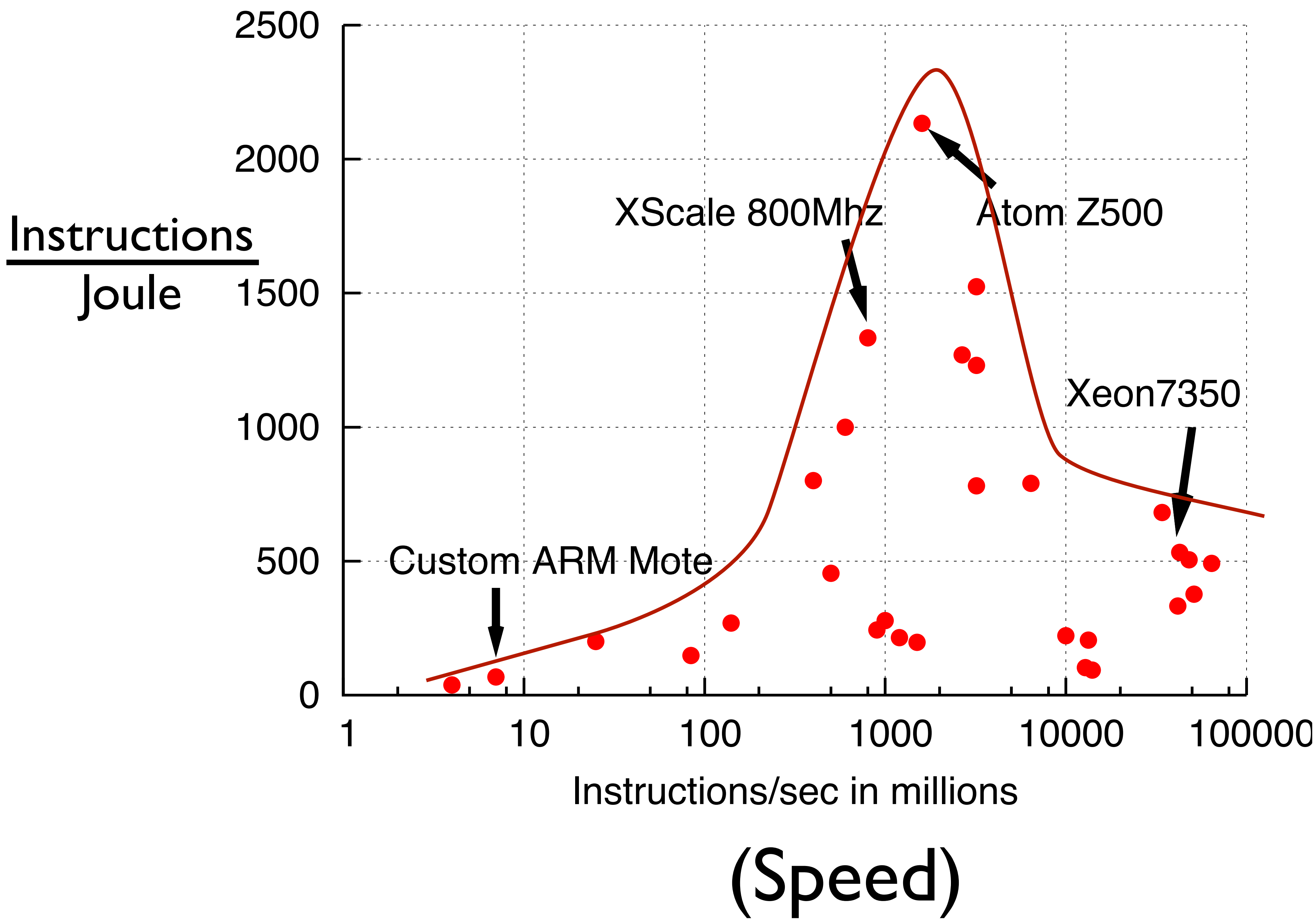
Leave it to industry



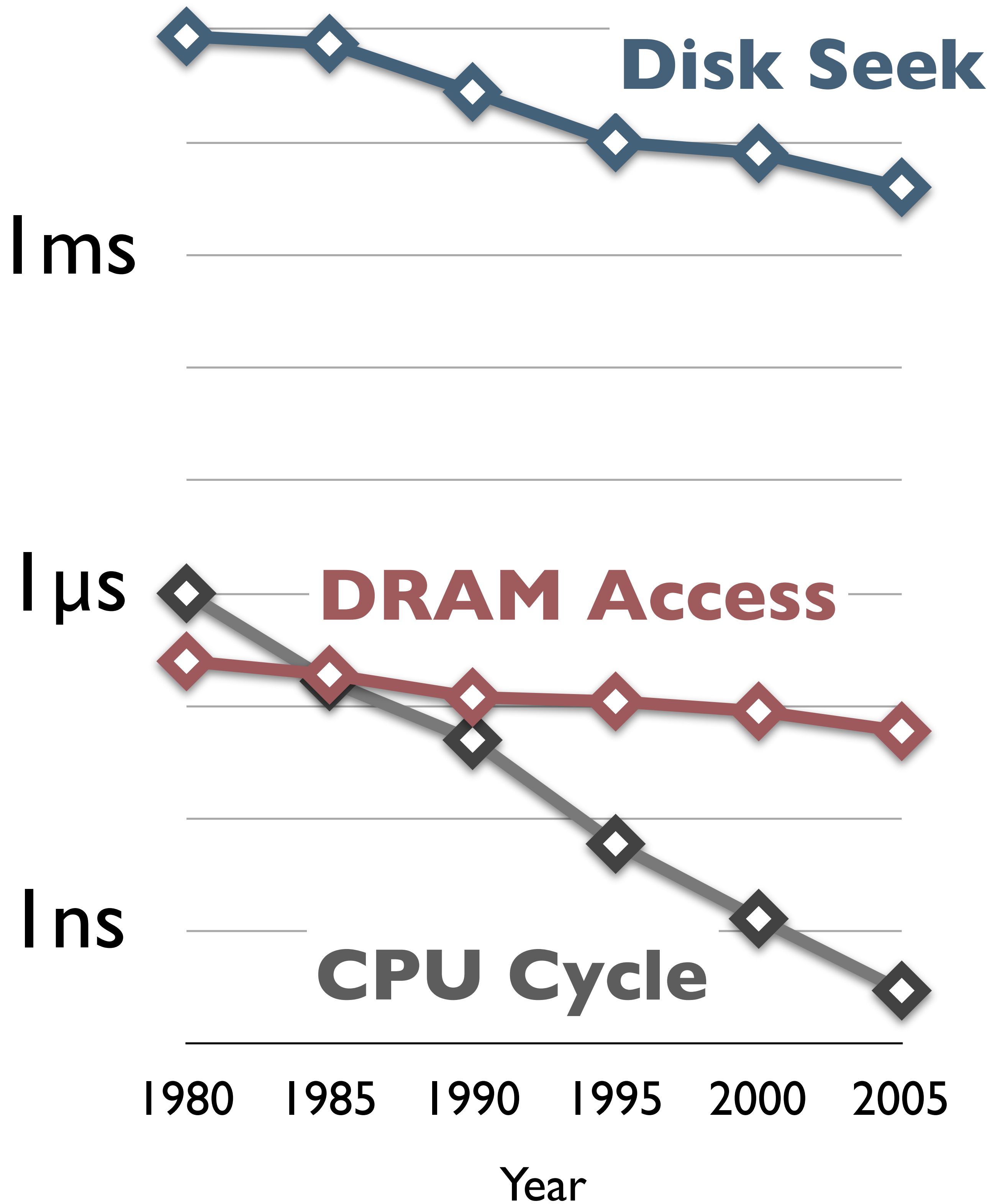

Infrastructure:
PUE
2005: 2–3
2012: ~1.1
Leave it to industry
100%
1000W
Servers
20%
Proportionality
750W
200W
100%
Efficiency
300W
FAWN

Gigahertz is not free

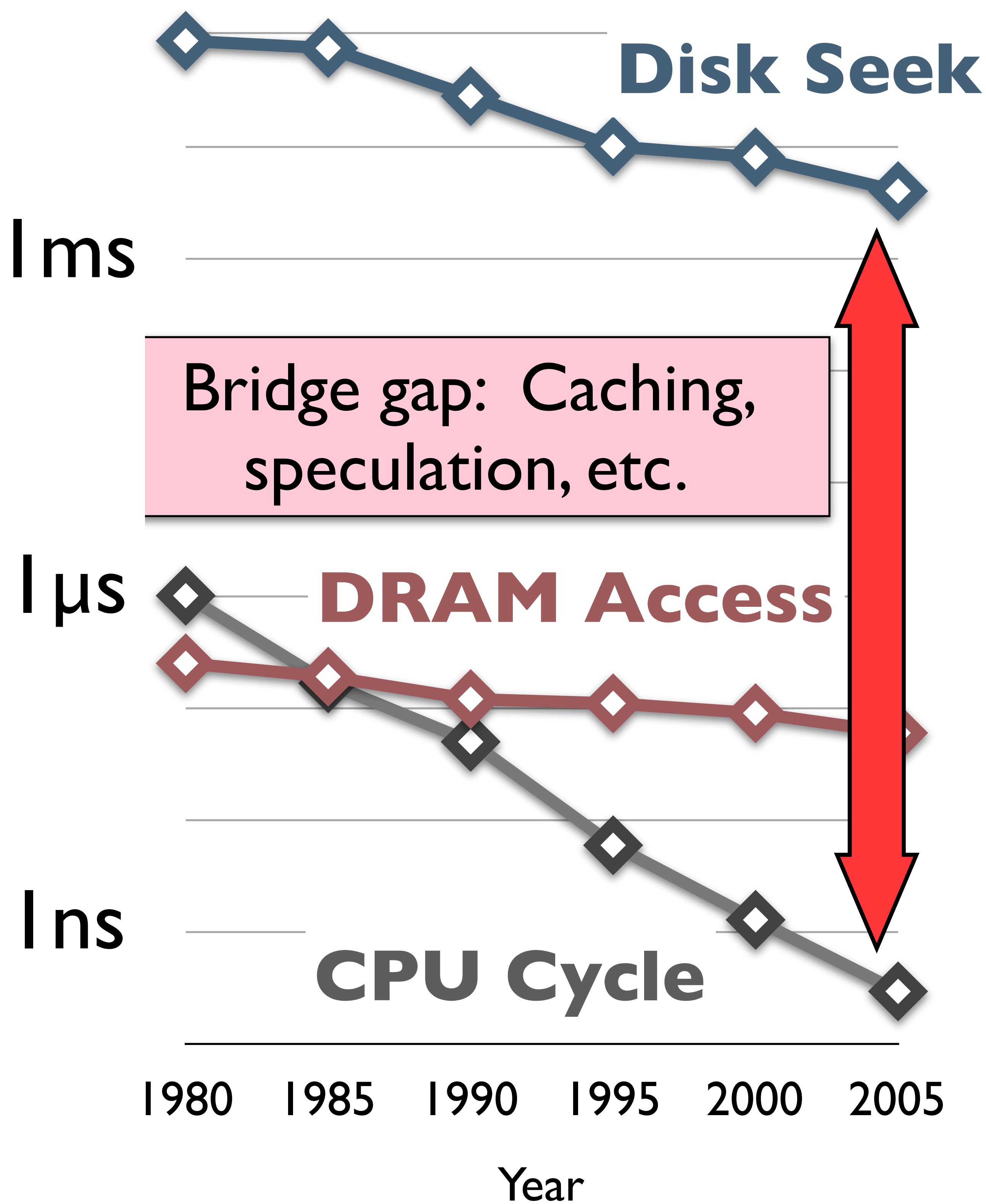
Speed and power calculated from specification sheets
Power includes “system overhead” (e.g., Ethernet)



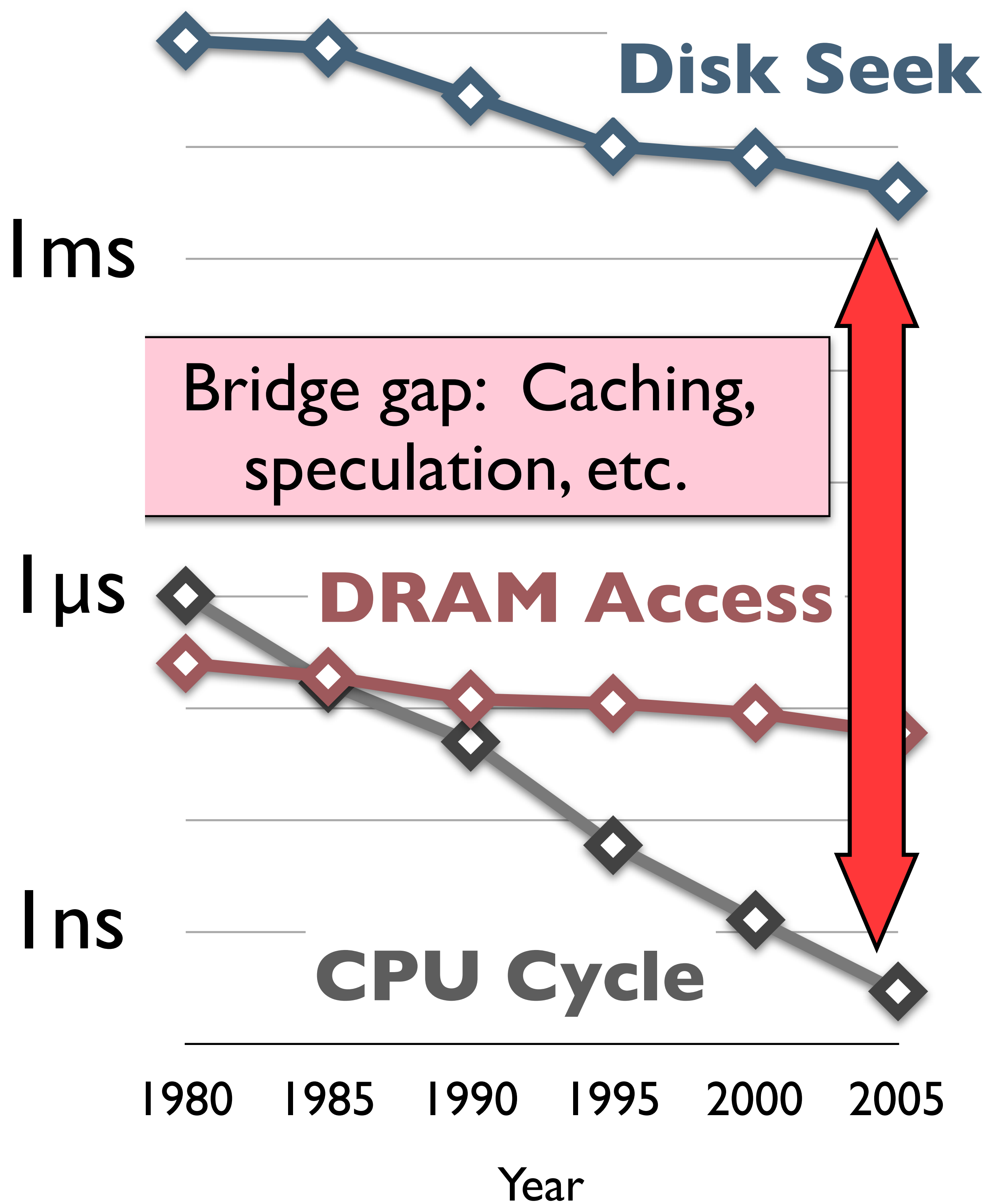
The Memory Wall



The Memory Wall

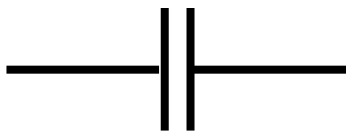


The Memory Wall

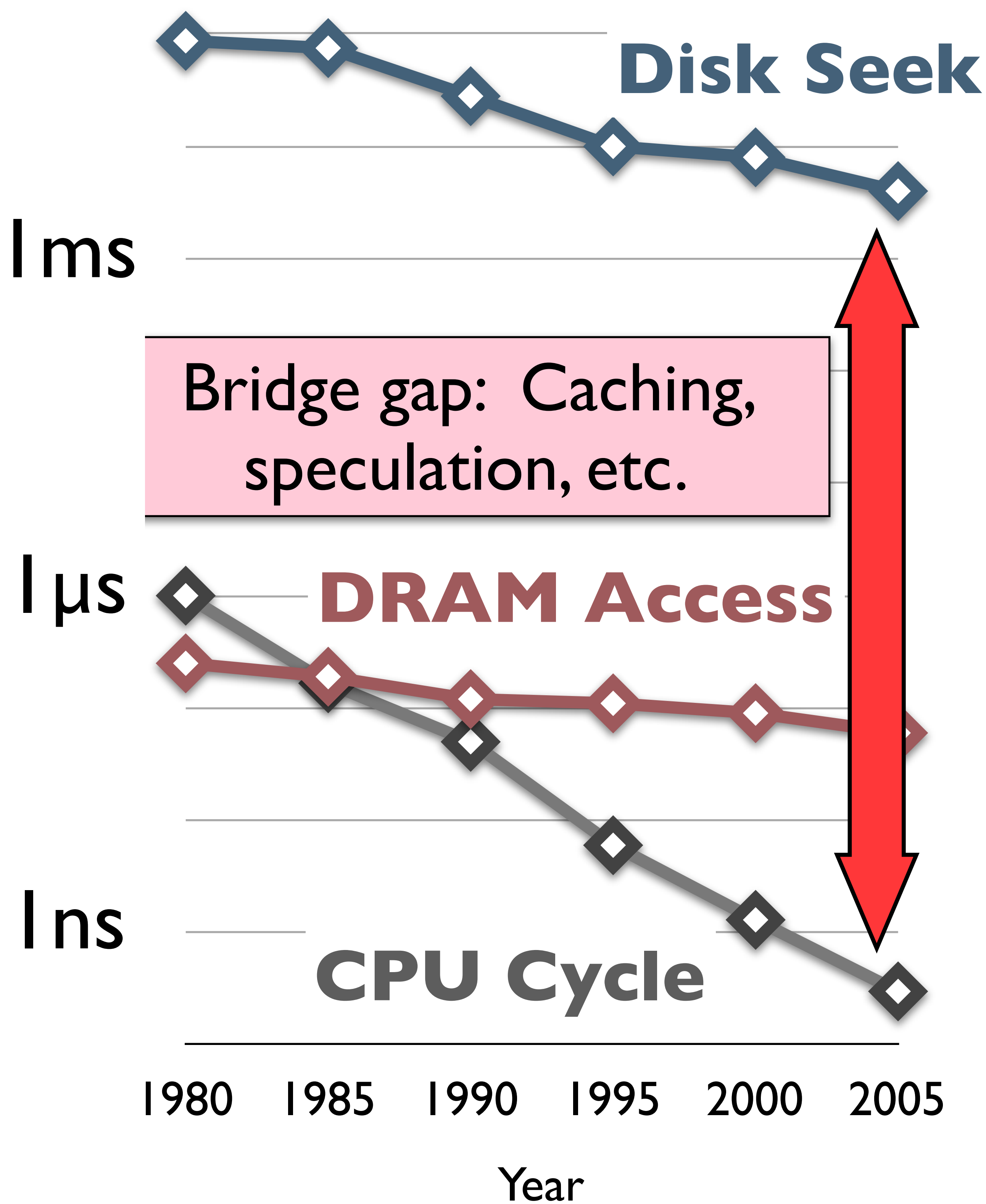


Transistors

Have the soul of a capacitor

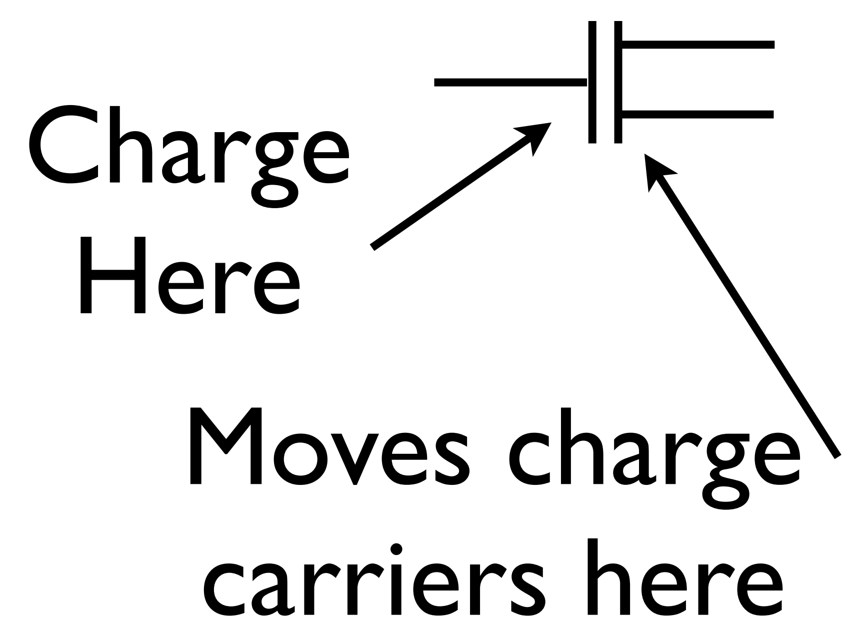


The Memory Wall



Transistors

Have the soul of a capacitor

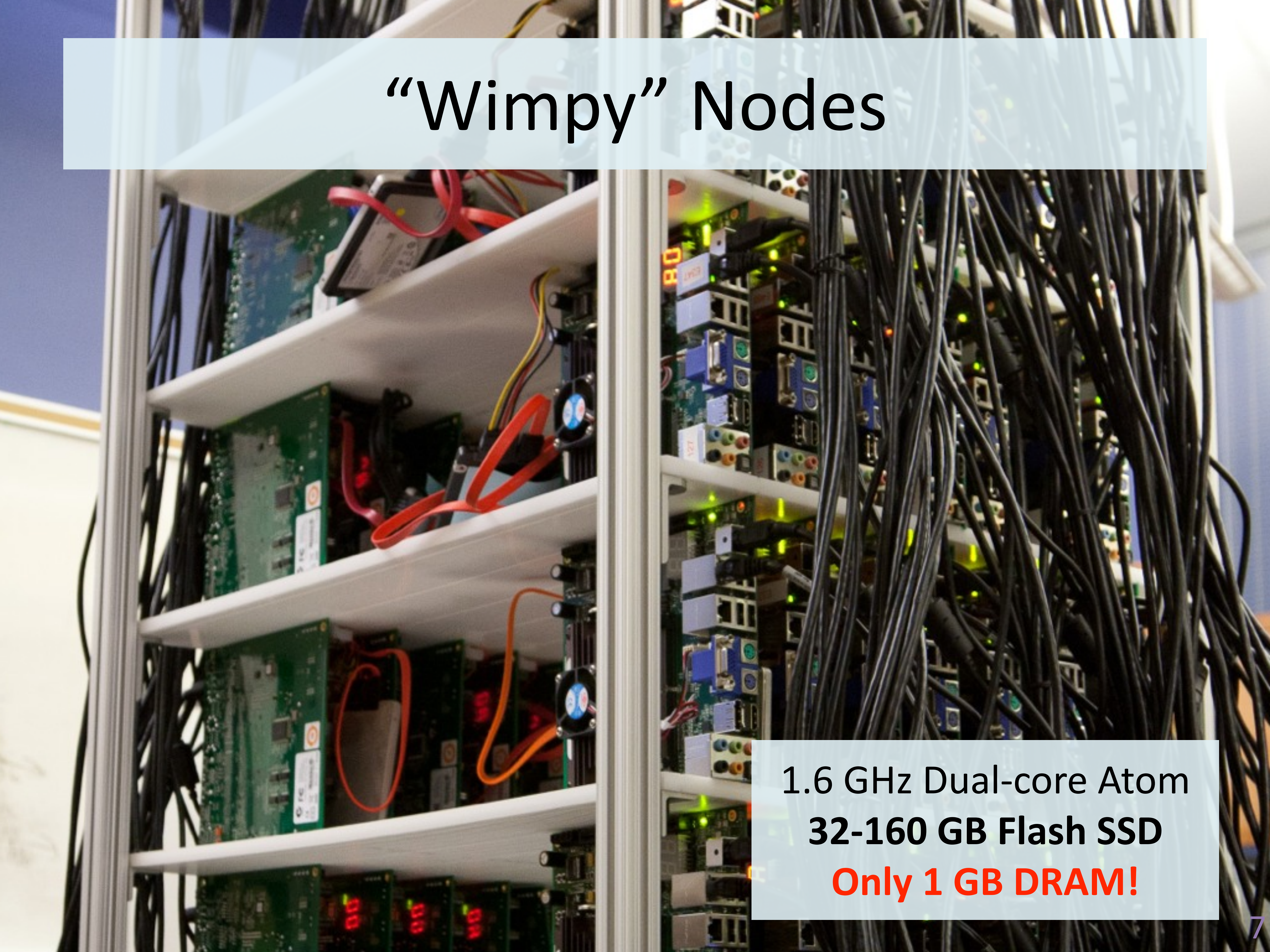


Which lets current flow

Gigahertz hurts

Remember:
Memory capacity costs you

“Wimpy” Nodes

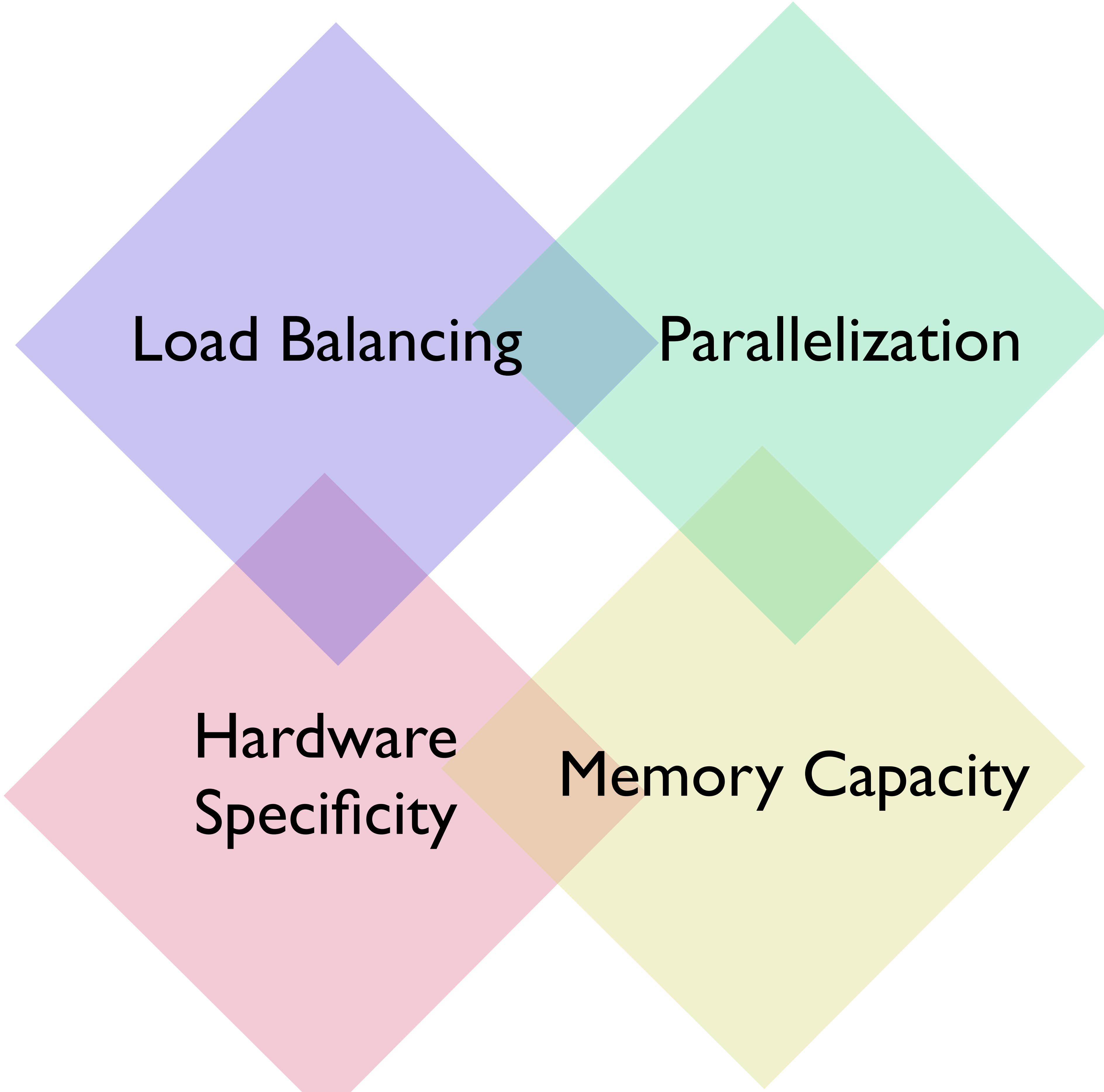


1.6 GHz Dual-core Atom

32-160 GB Flash SSD

Only 1 GB DRAM!

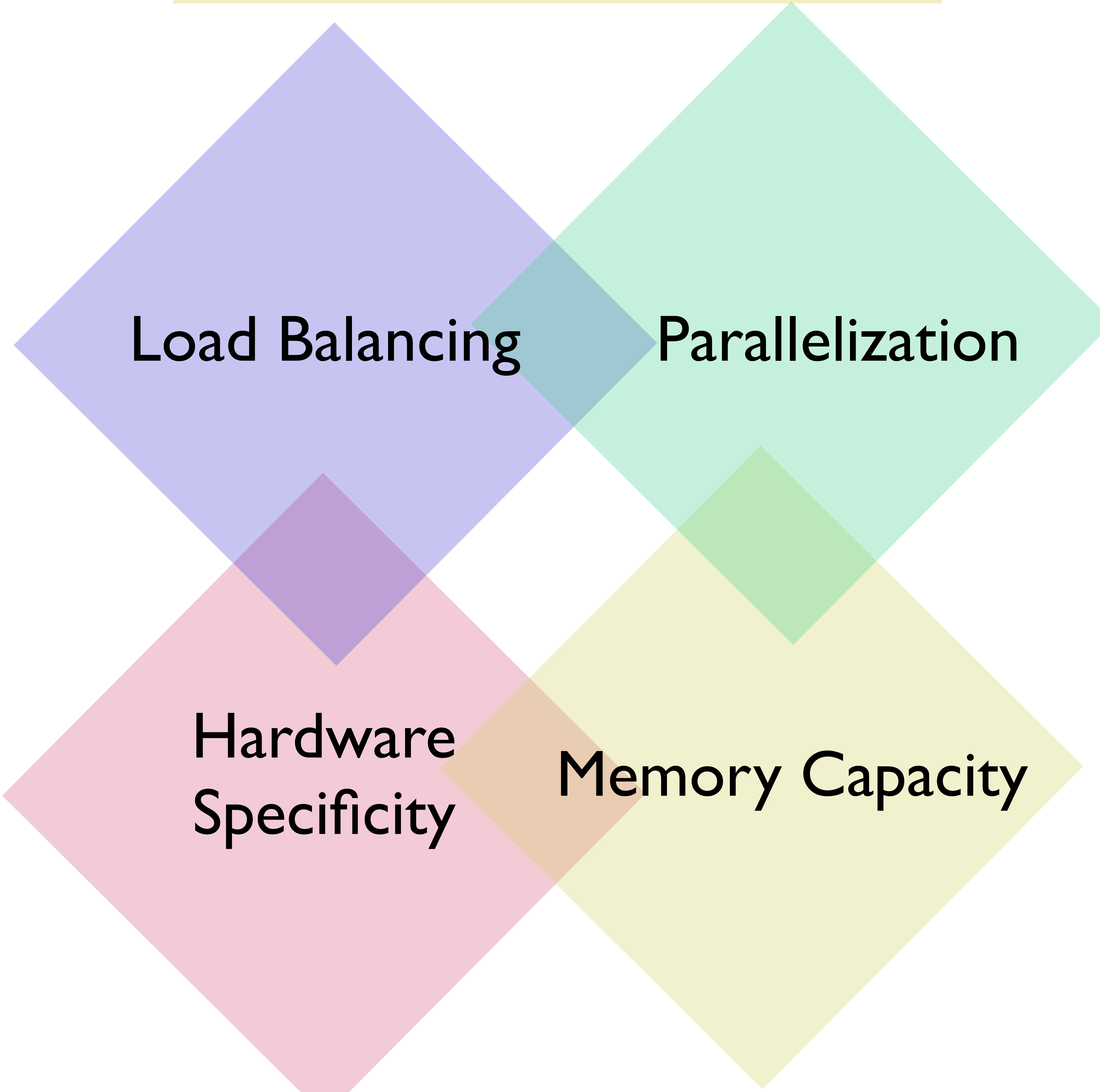
"Each decimal order of magnitude increase in parallelism requires a major redesign and rewrite of parallel code" - Kathy Yelick



Bigger Clusters

Wimpy Nodes

The FAWN Quad of Pain

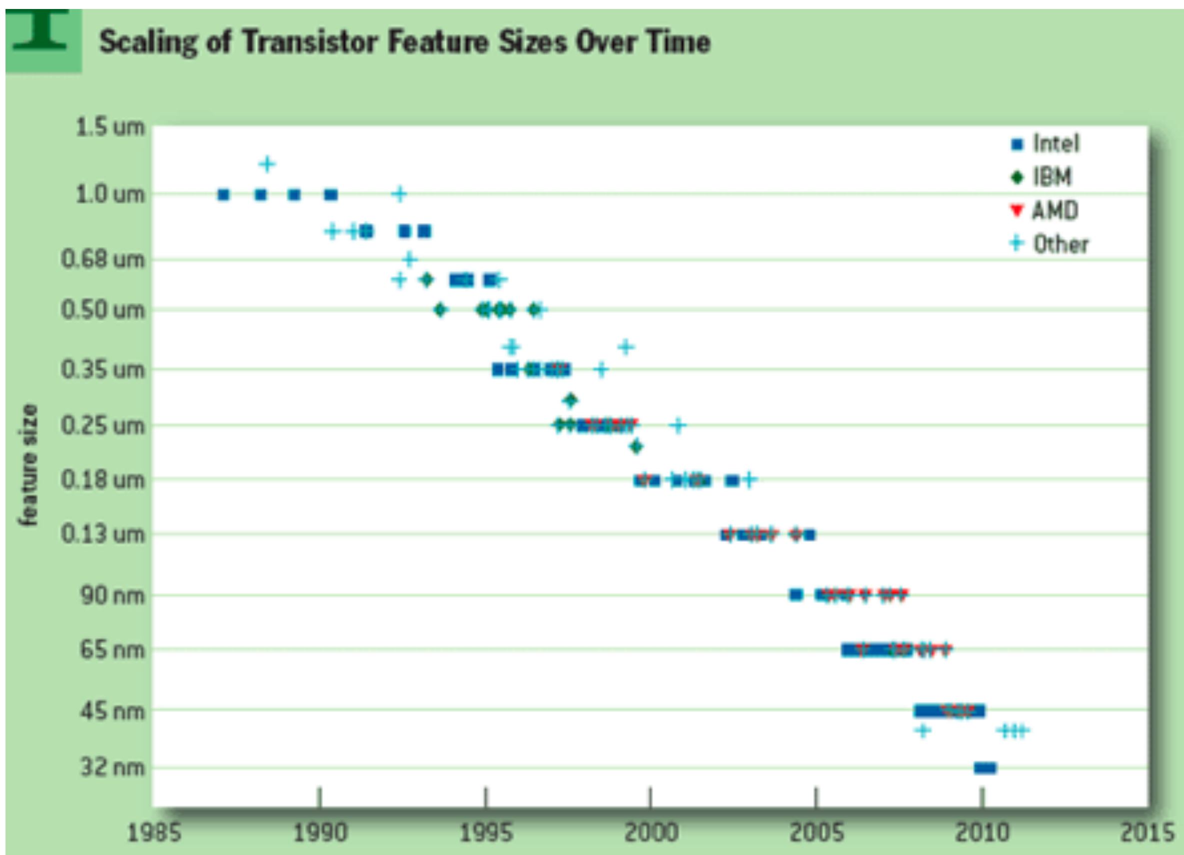


Bigger Clusters

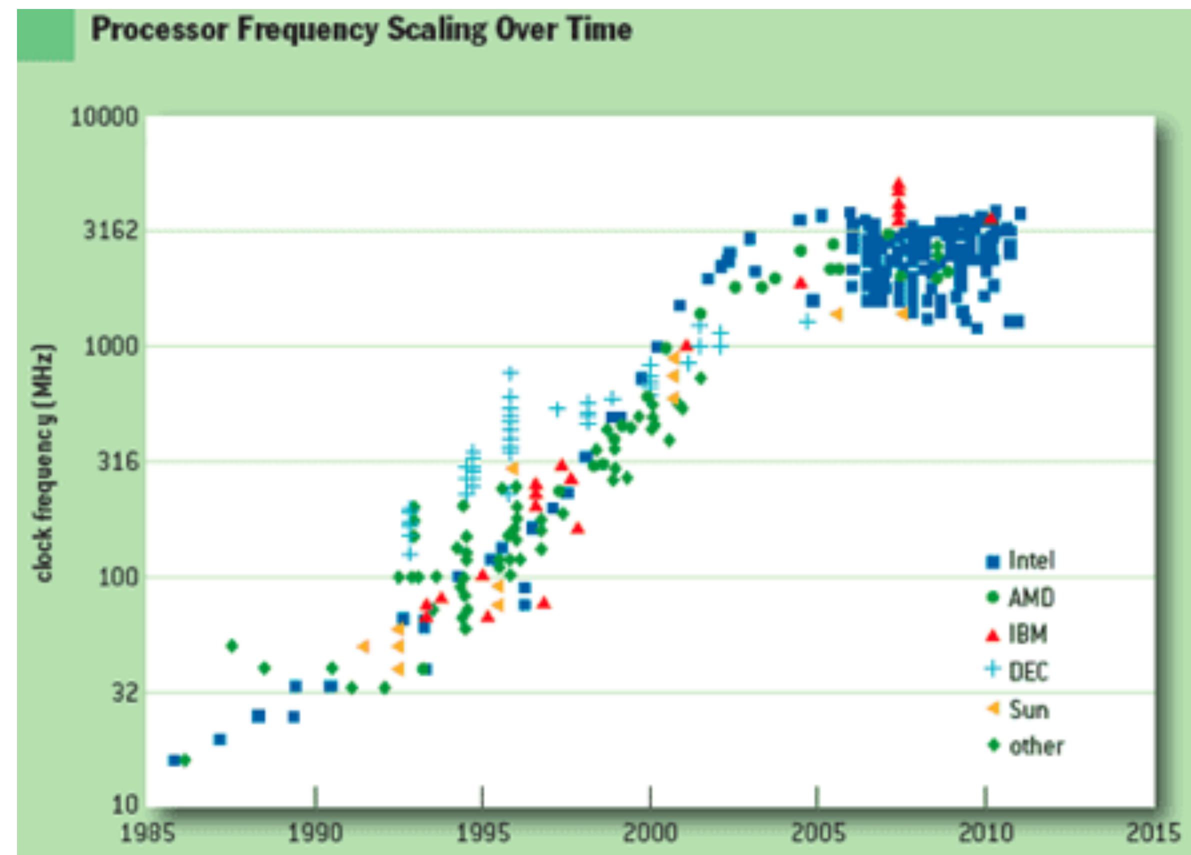
Wimpy Nodes

It's not just masochism

Moore



Dennard



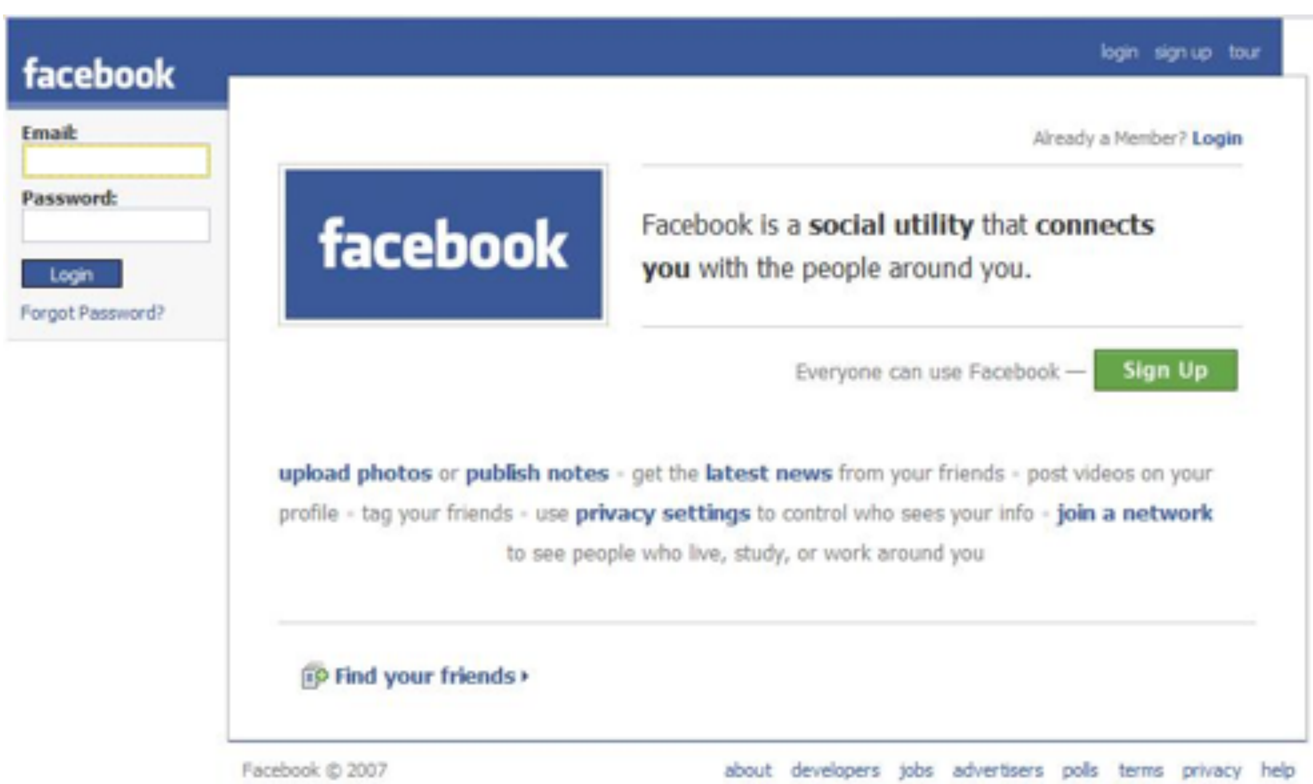
(Figures from Danowitz, Kelley, Mao, Stevenson, and Horowitz: CPU DB)

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

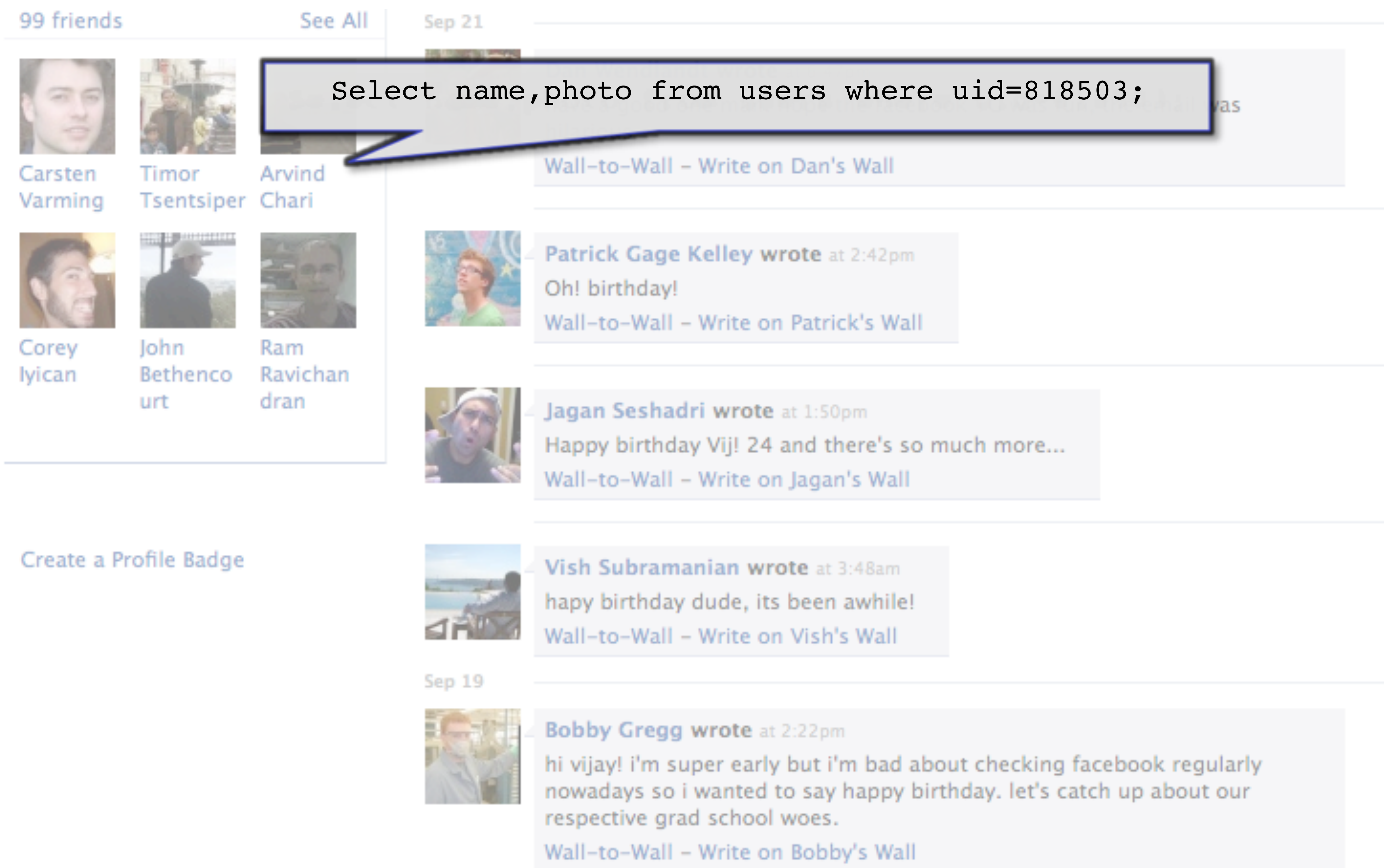


Small record, random access

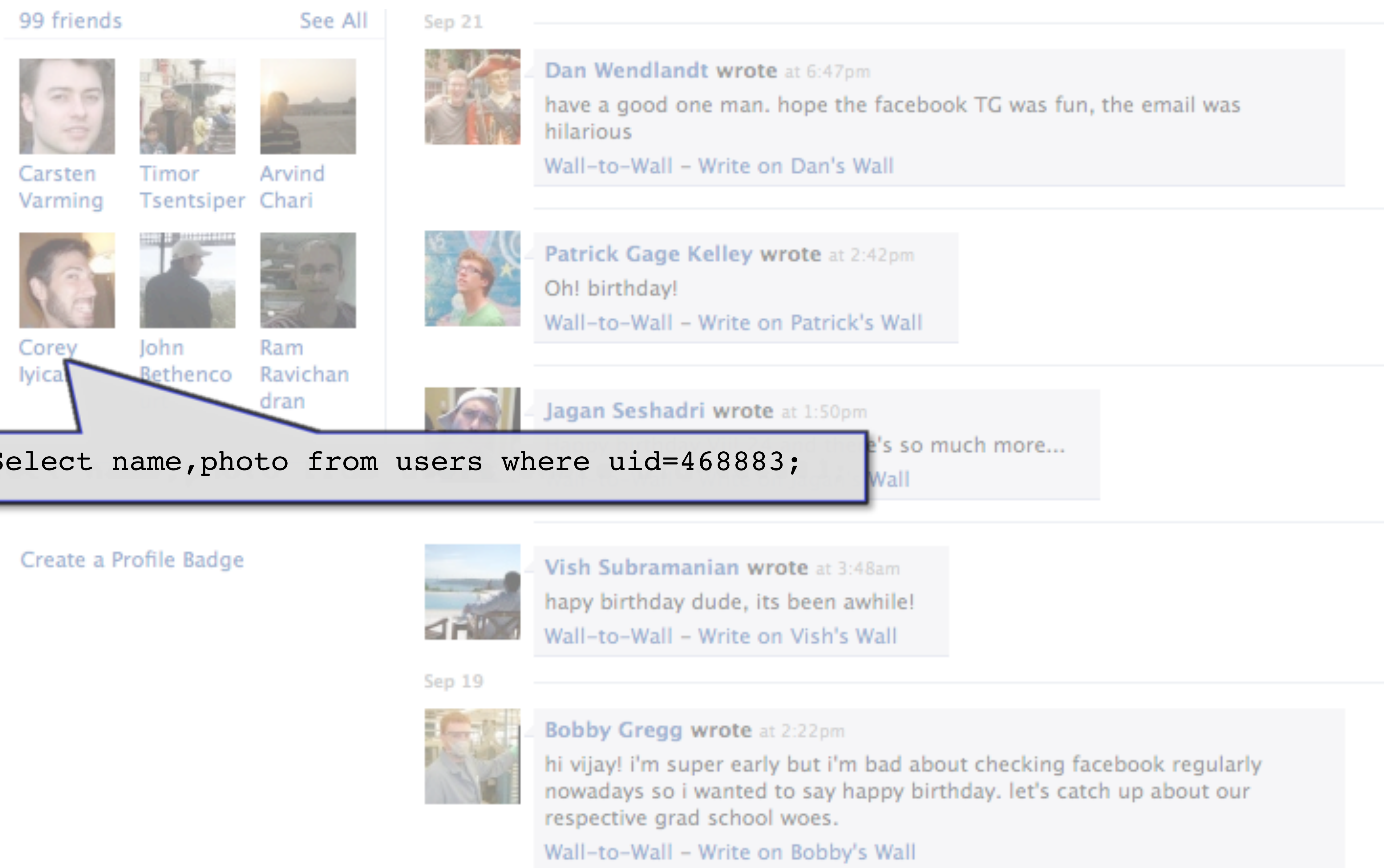
Small record, random access

```
Select name,photo from users where uid=513542;
```

Small record, random access

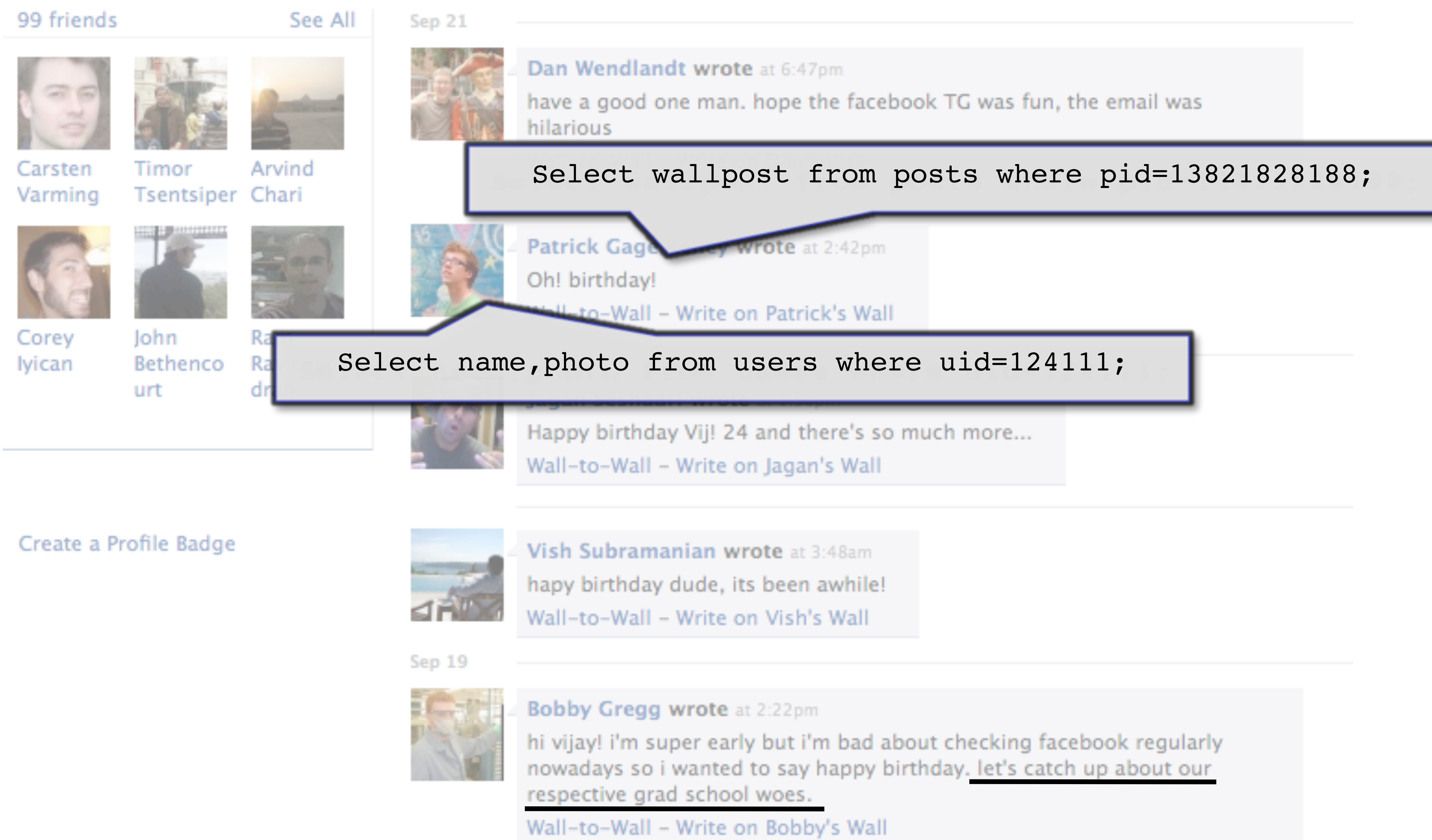


Small record, random access




```
Select name,photo from users where uid=468883;
```

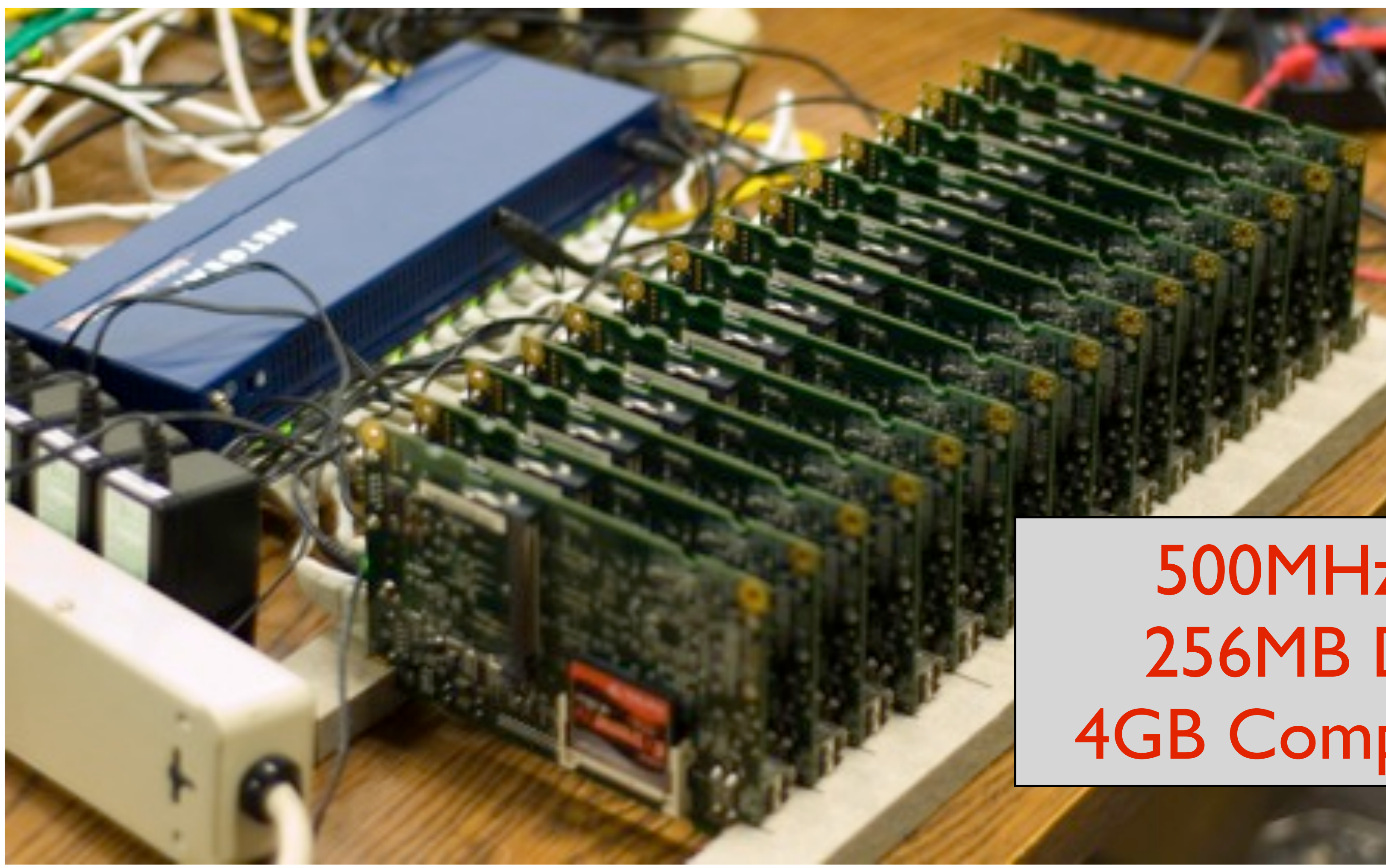
Small record, random access



Small record, random access

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

Goal: improve **Queries/Joule**



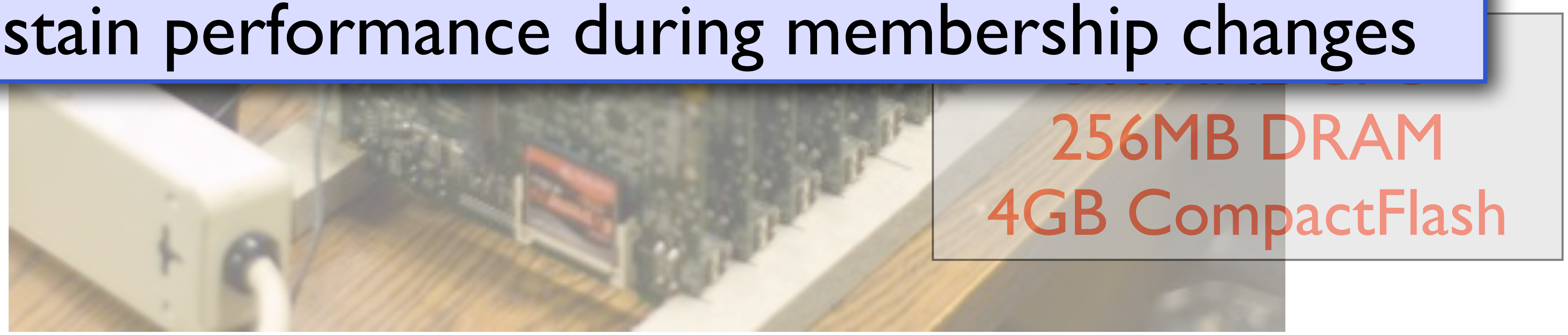
500MHz CPU
256MB DRAM
4GB CompactFlash

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

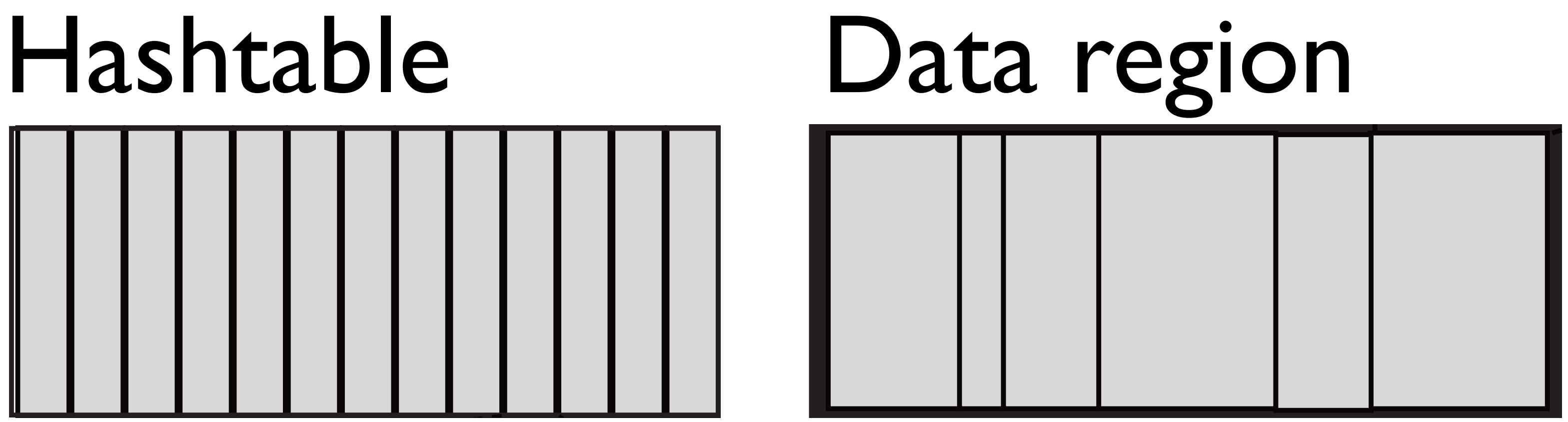
Goal: improve **Queries/Joule**

Unique Challenges:

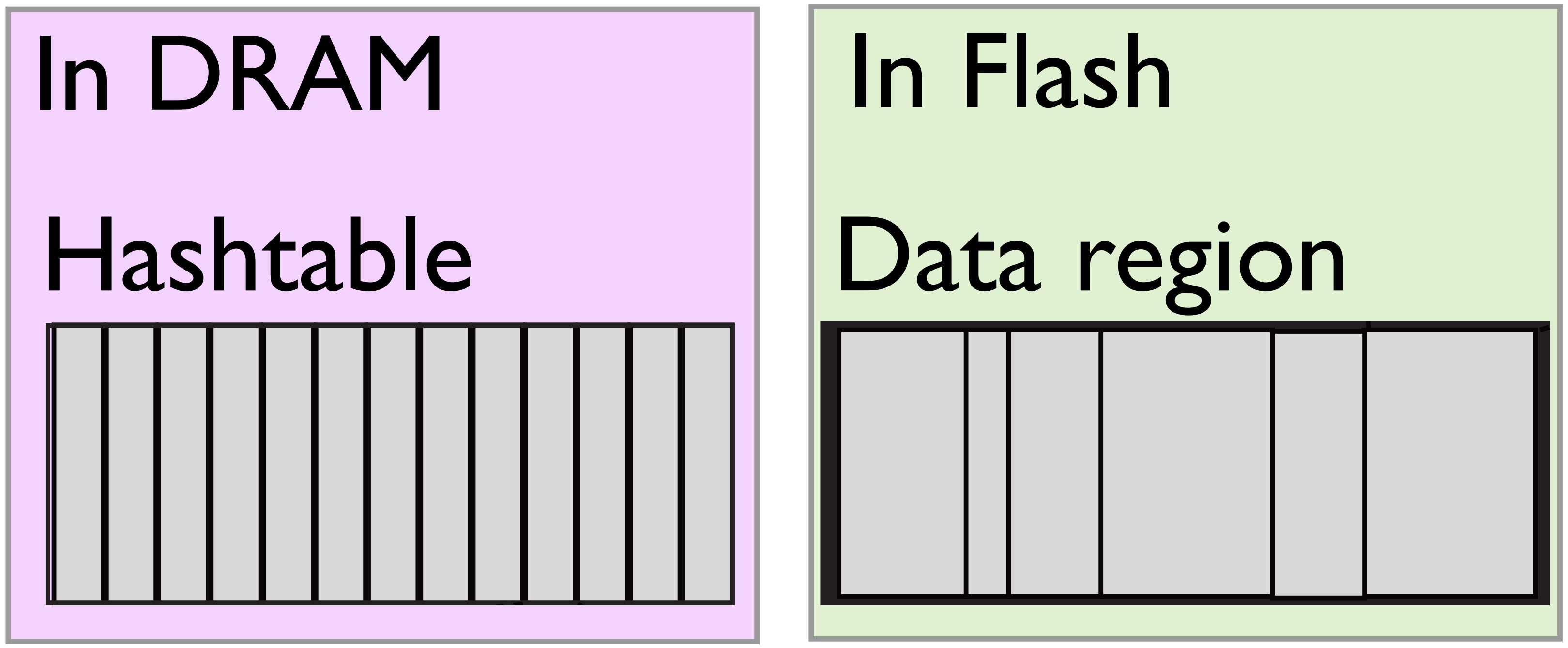
- Wimpy CPUs, limited DRAM
- Flash poor at small random writes
- Sustain performance during membership changes



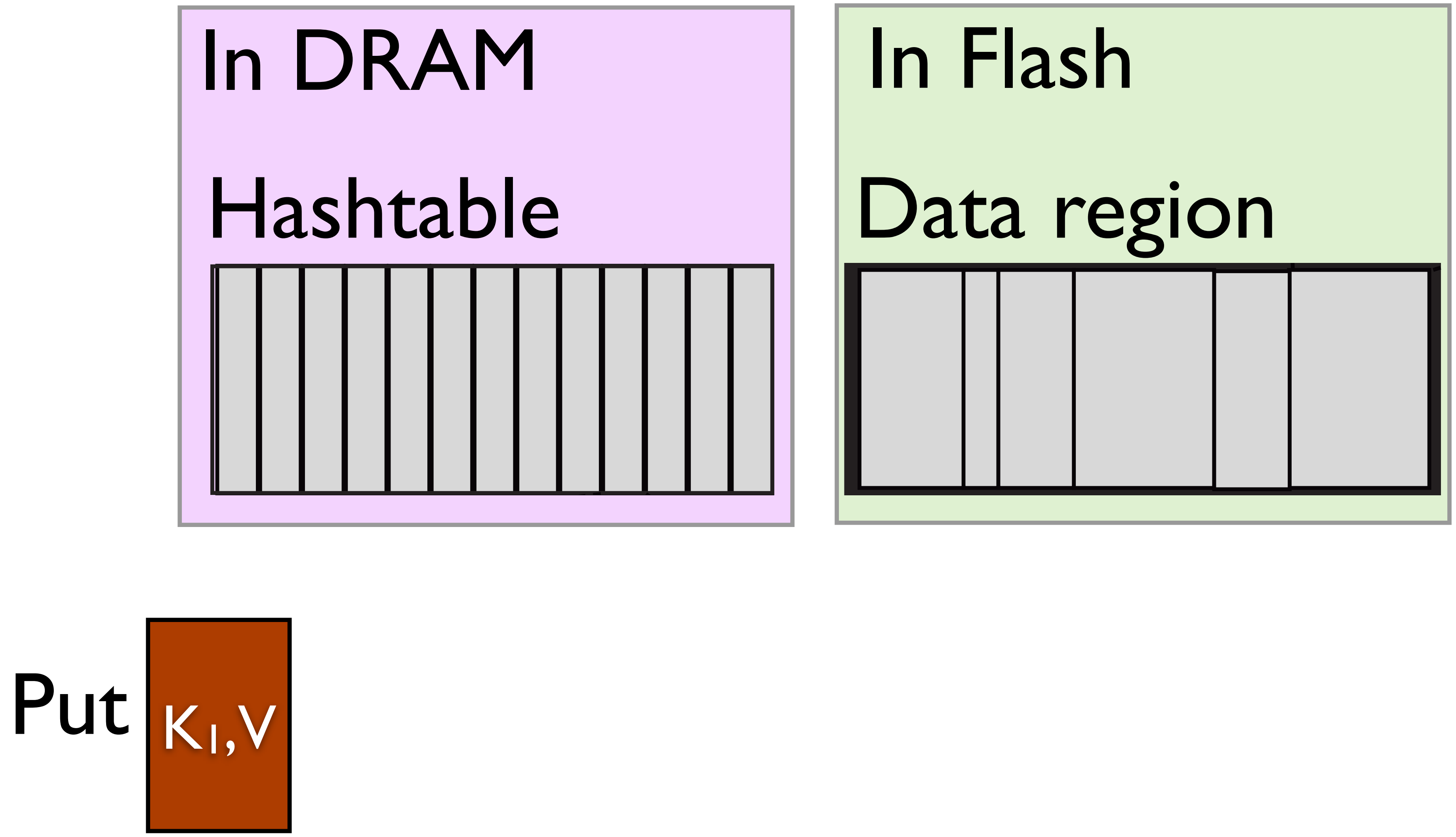
Avoiding random writes



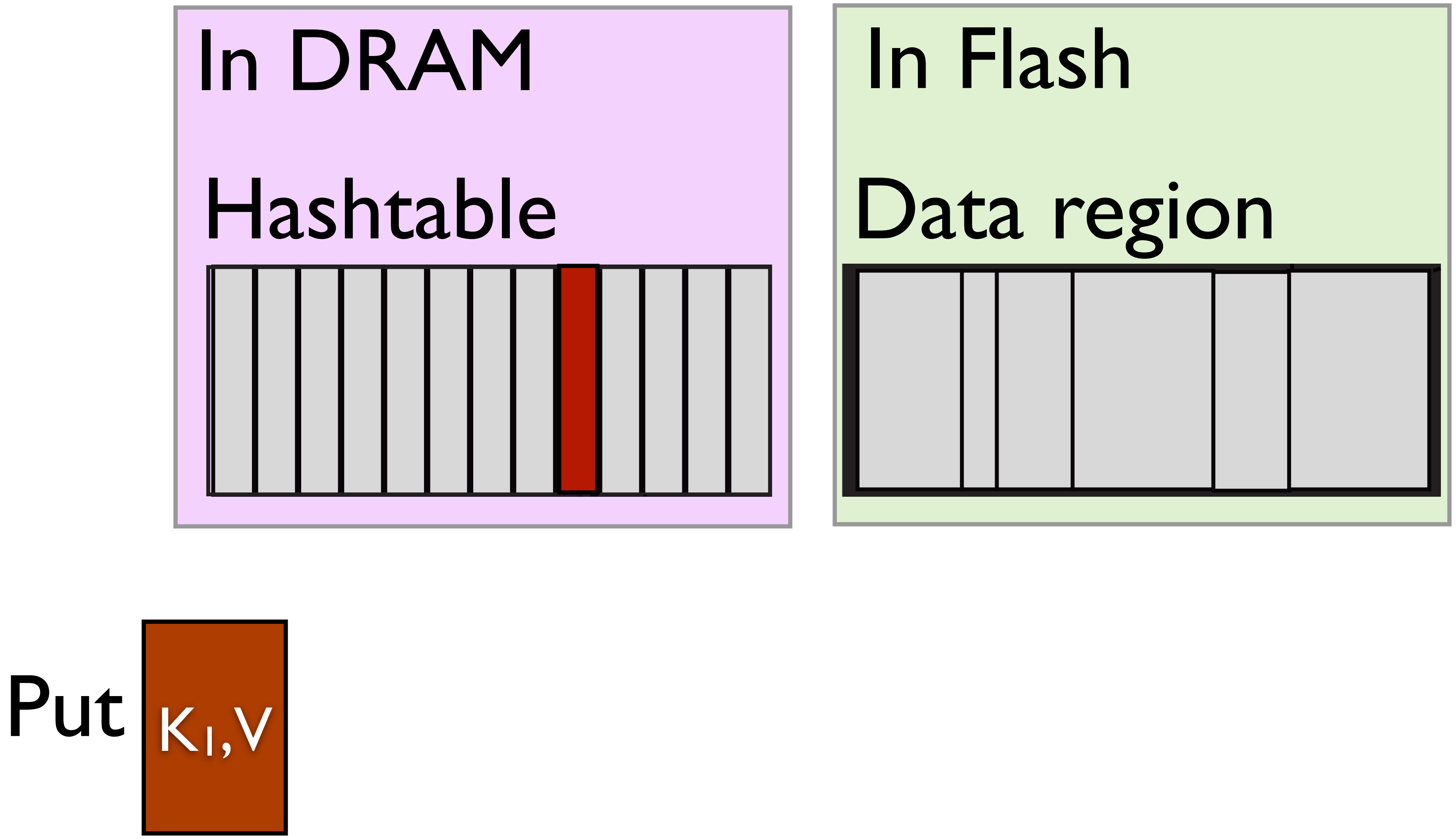
Avoiding random writes



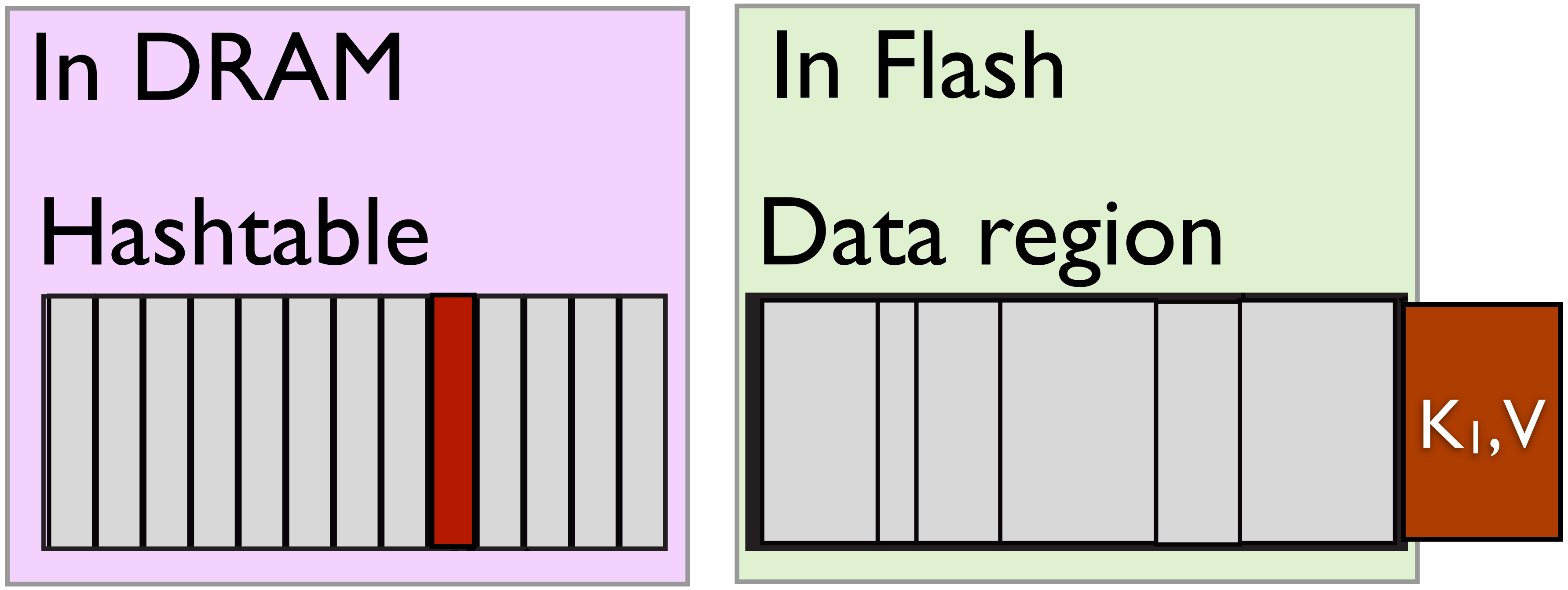
Avoiding random writes



Avoiding random writes

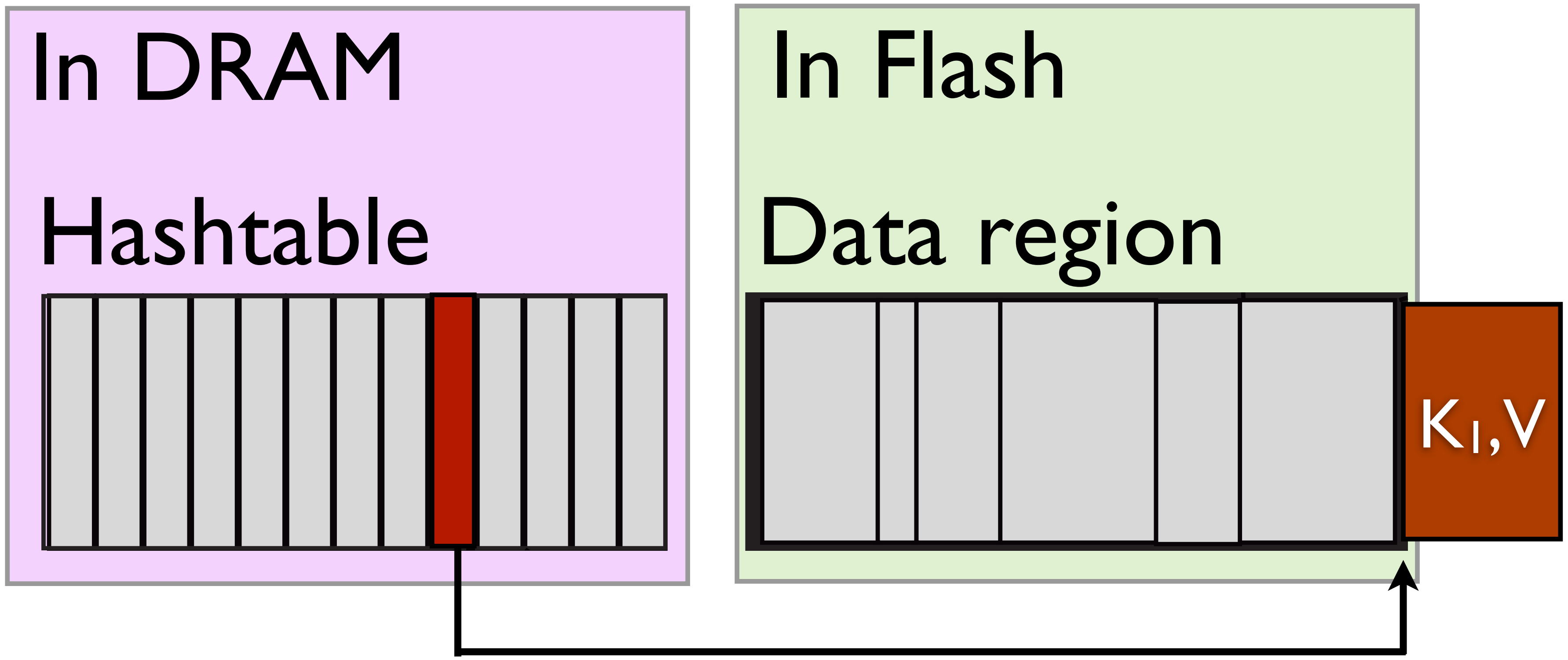


Avoiding random writes



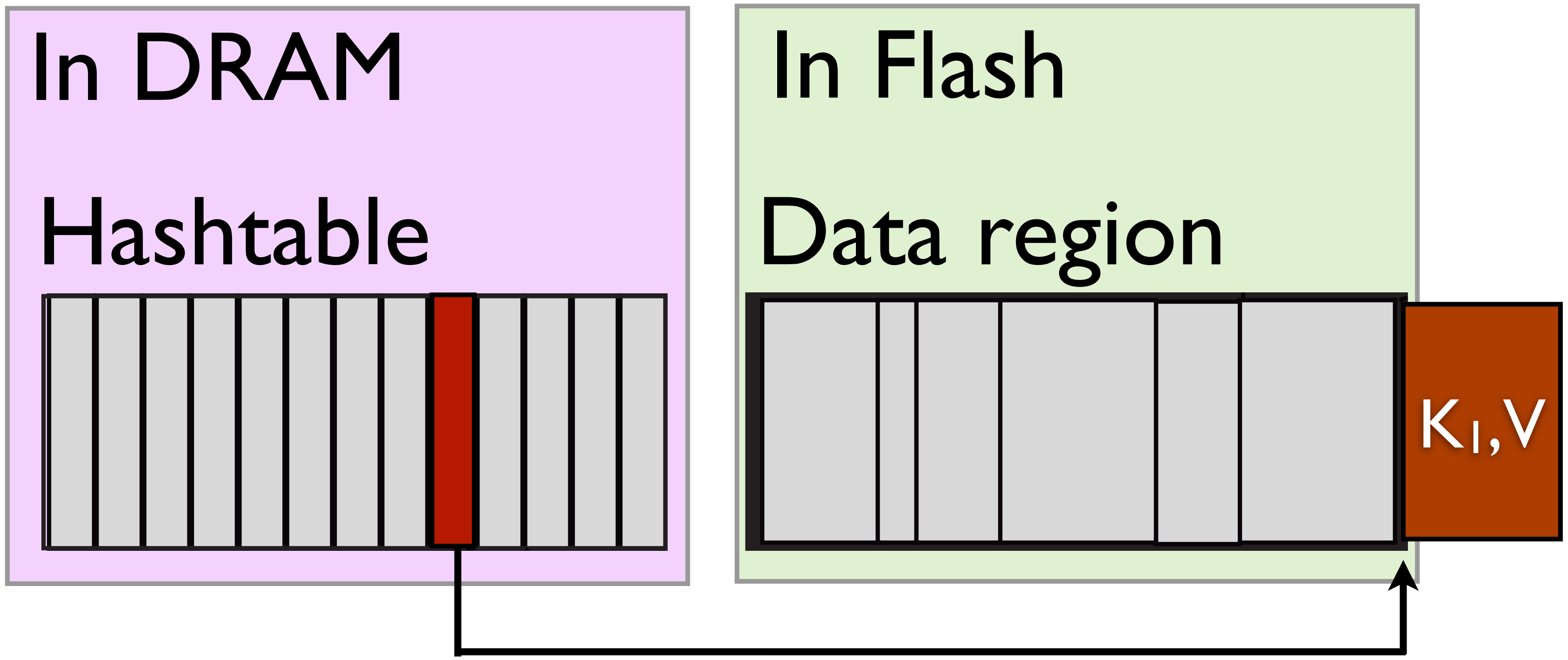
Put

Avoiding random writes



Put

Avoiding random writes



Put

All writes to Flash are sequential

Research Example

- Developed DRAM-efficient system to find location on flash
  - ("Partial-key hashing") 2008-9
- We've continued this since then:
  - Partial-key cuckoo hashing 2011
  - Optimistic concurrent cuckoo hashing 2012

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-performance, 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

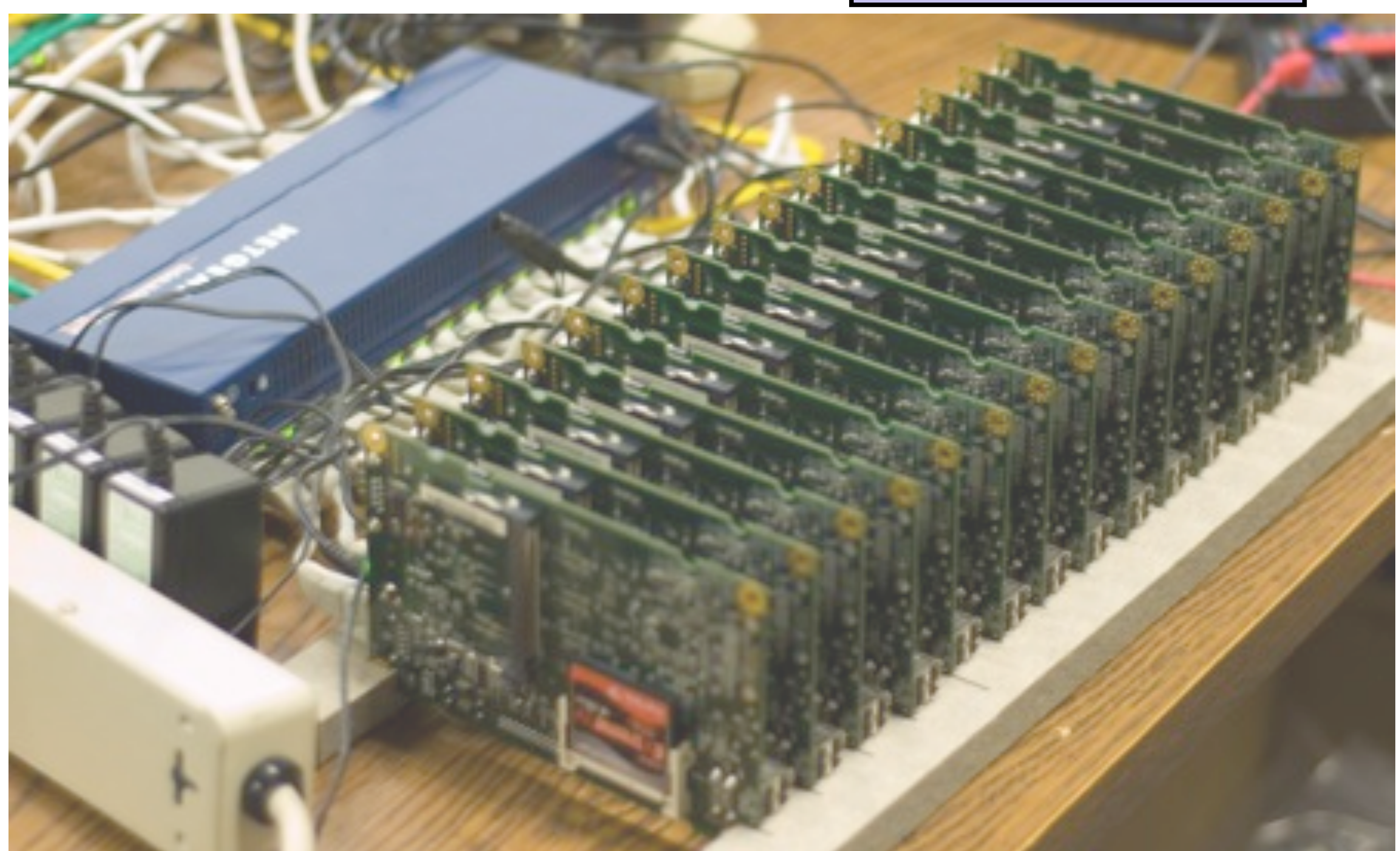
FAWN-DS FAWN-KV Small Cache Cuckoo

Fawn-DS

Fawn-KV

Fawn-DS

Fawn-DS



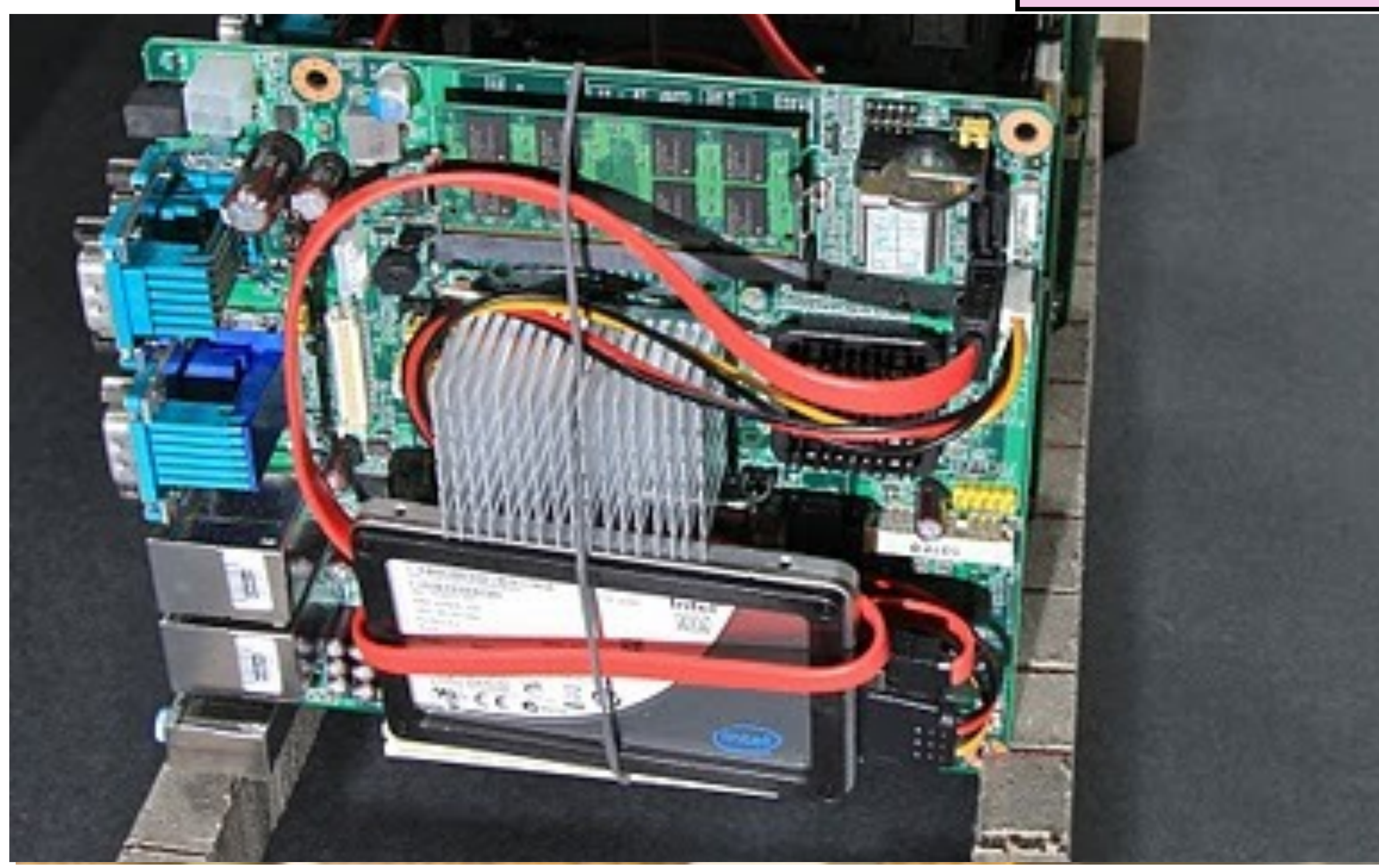
backend store
hyper-optimized
for low DRAM
and large flash

Fawn-KV

SILT

SILT

SILT



Systems begat algorithms:

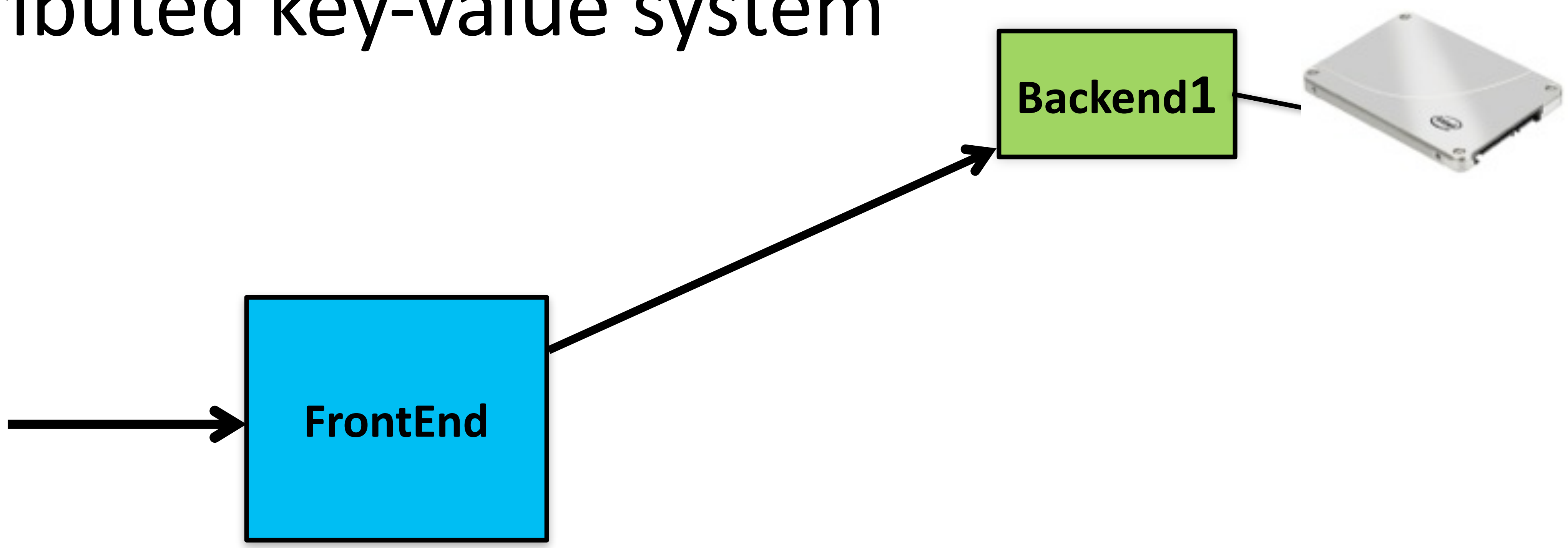
“Practical Batch-Updtable External Hashing with Sorting”

H. Lim et al., ALENEX 2012

(Recently heard that Bing uses several state-of-the-art, memory-efficient indexes)

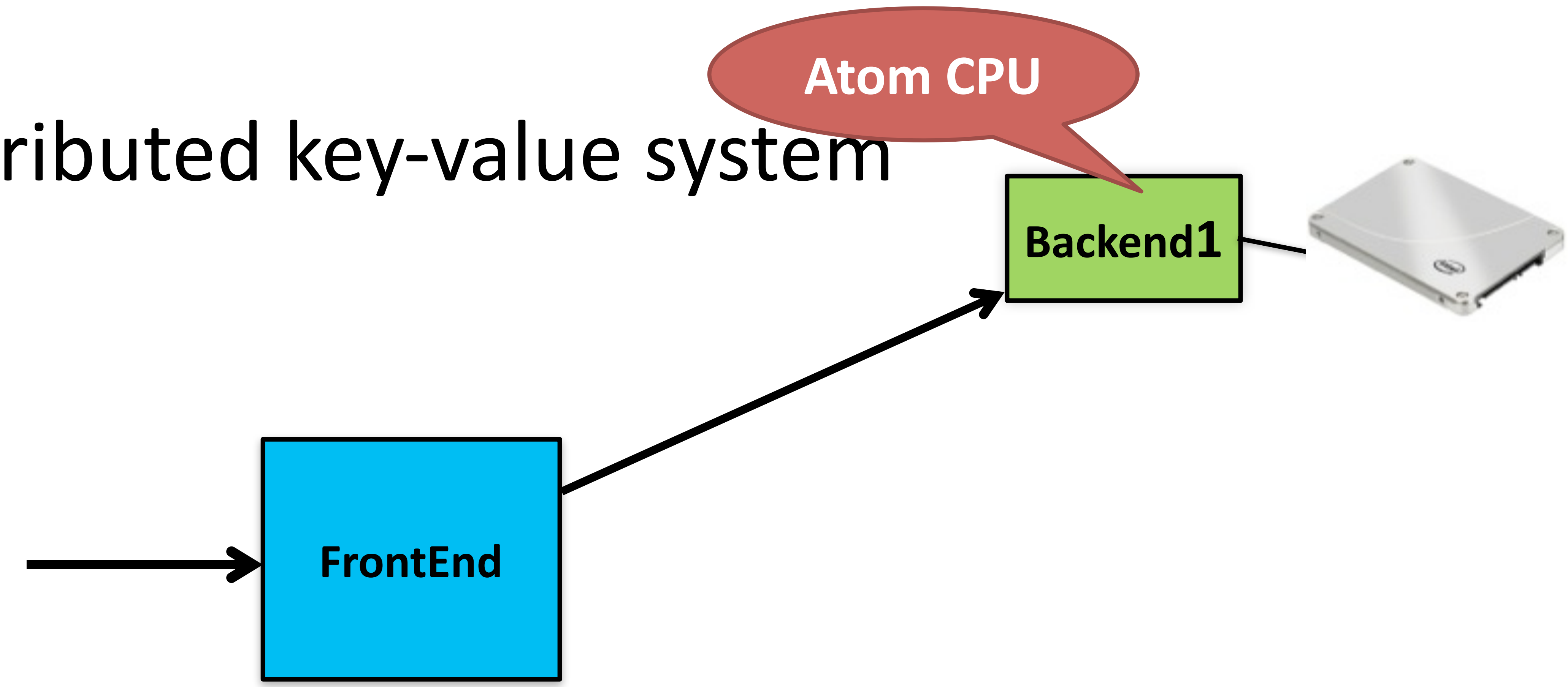
And now… Load imbalance

- Distributed key-value system



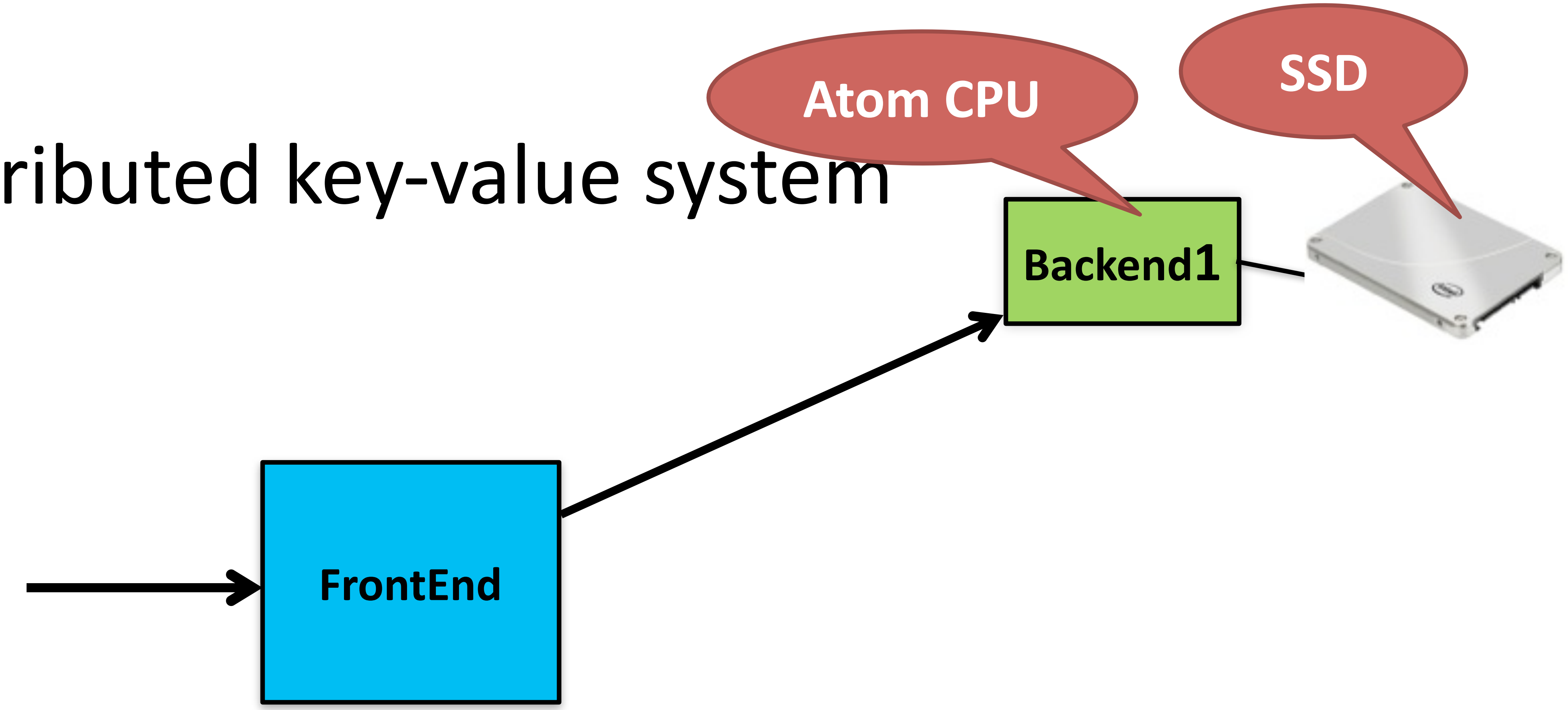
And now... Load imbalance

- Distributed key-value system



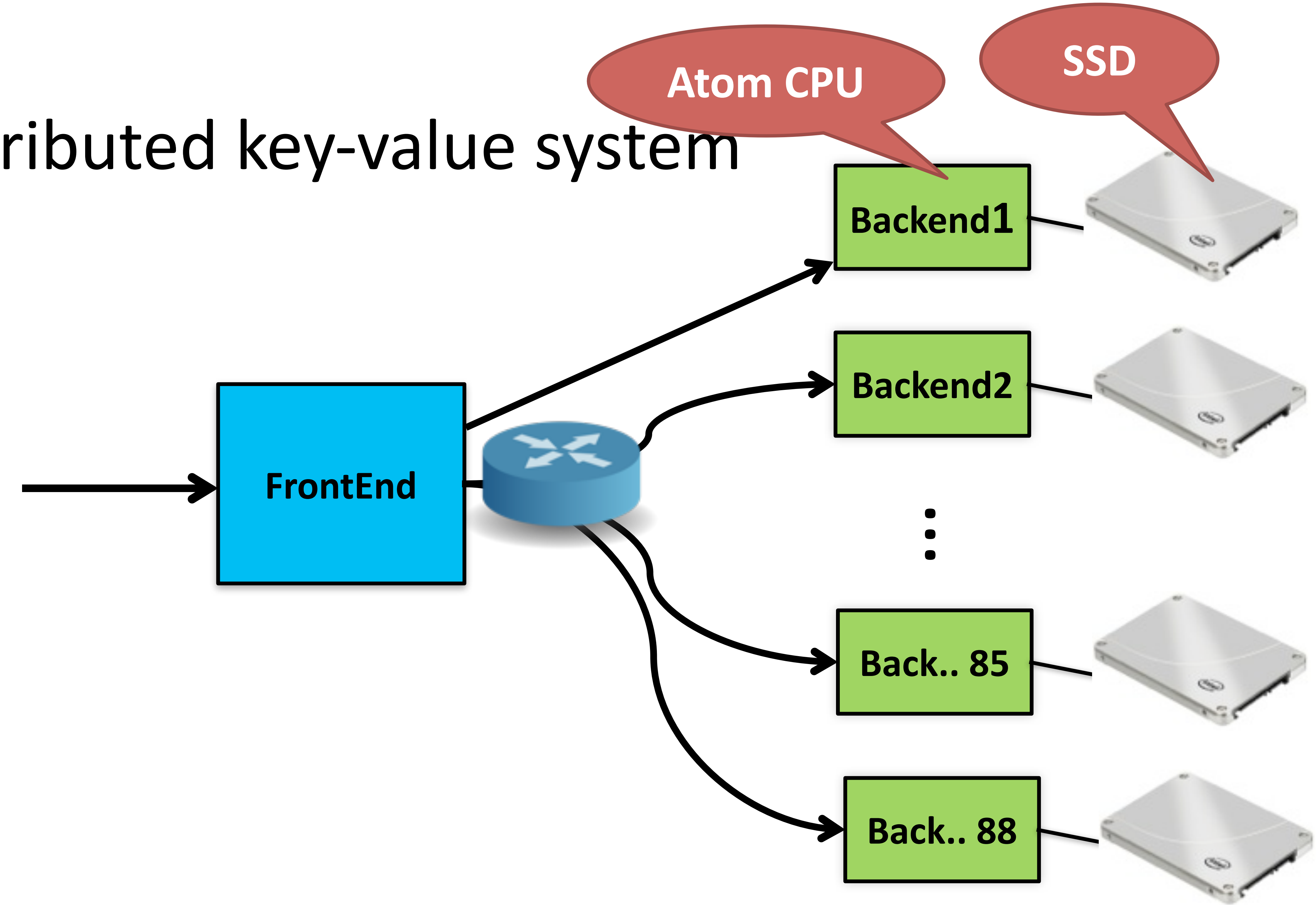
And now... Load imbalance

- Distributed key-value system



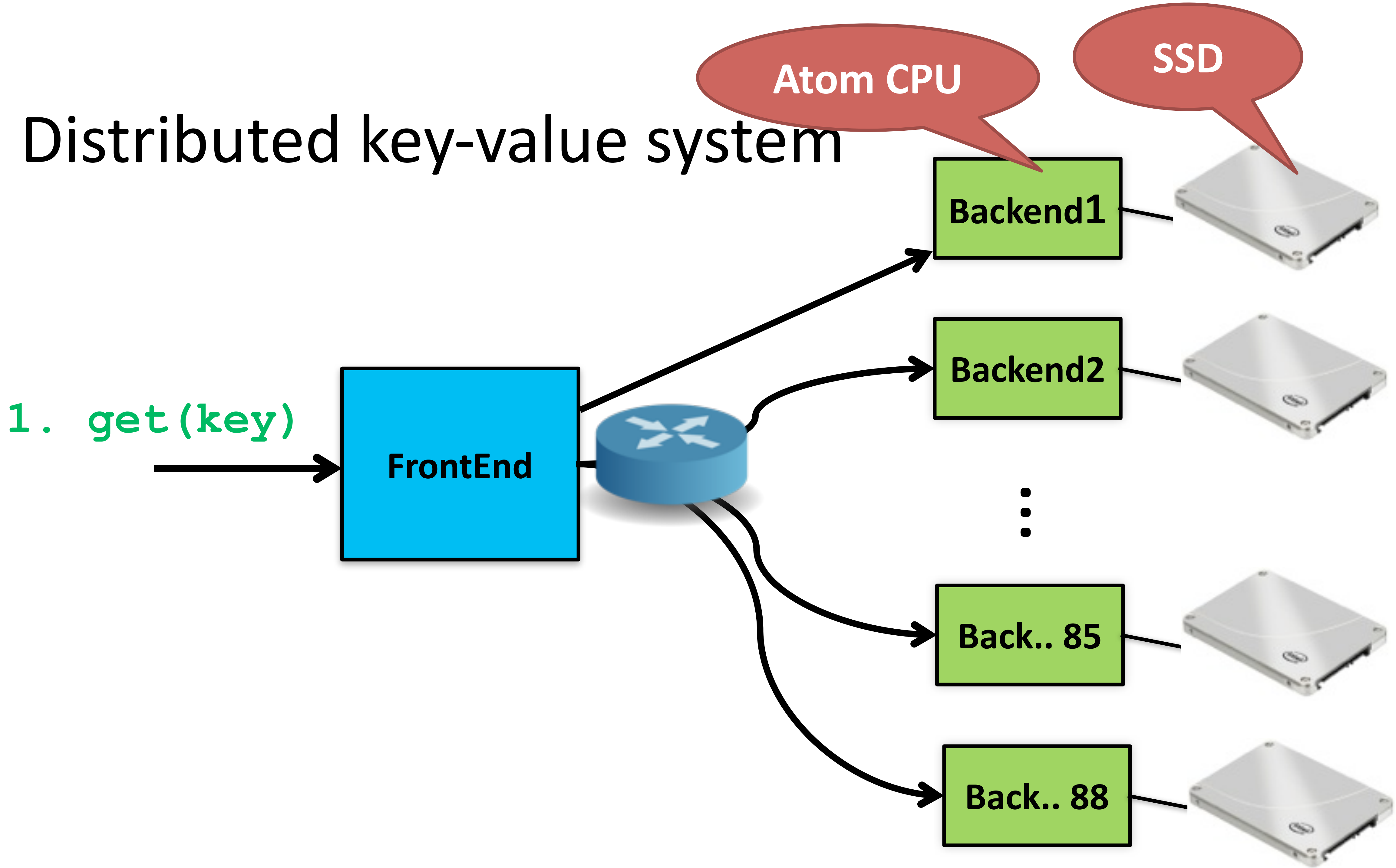
And now… Load imbalance

- Distributed key-value system



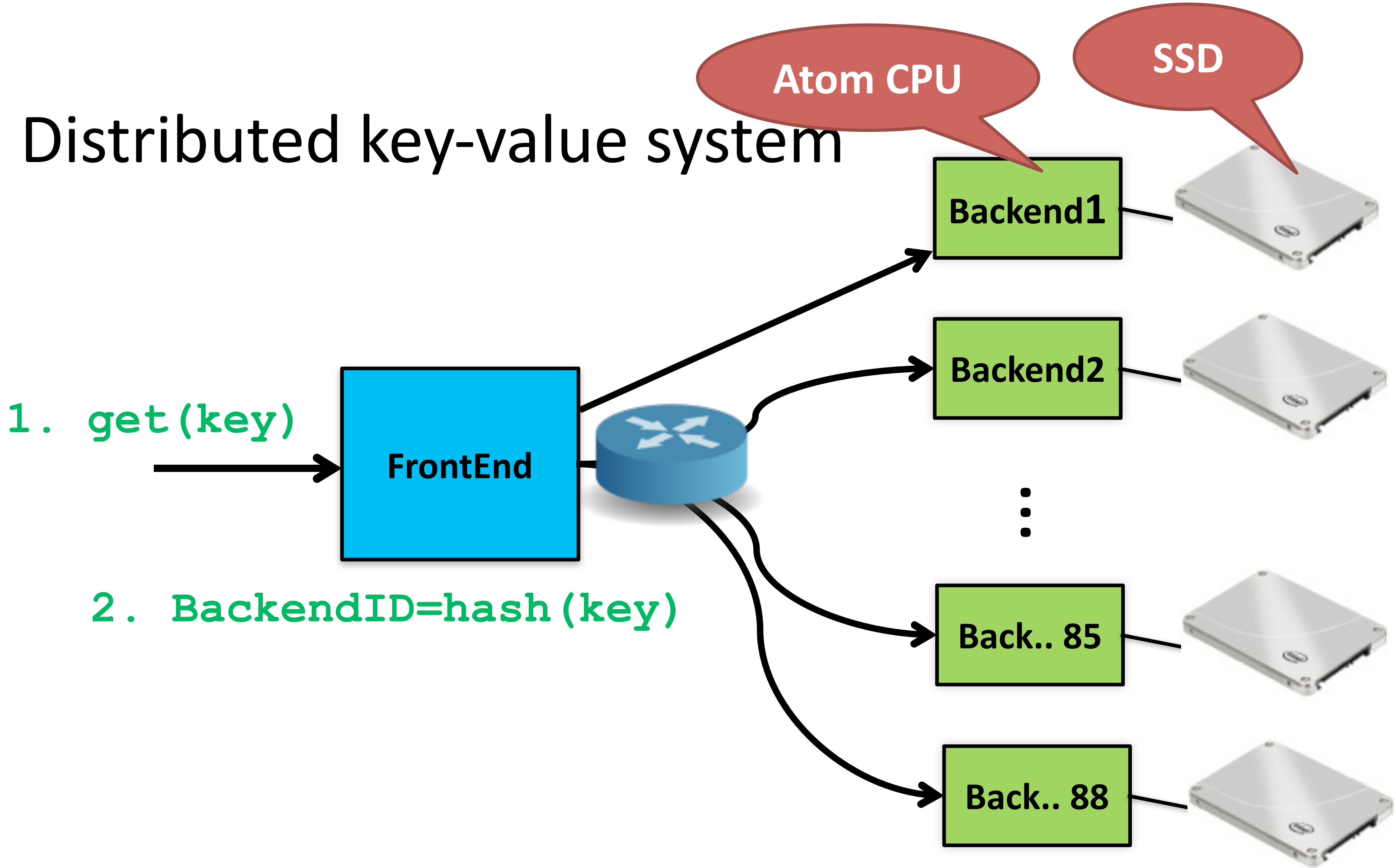
And now... Load imbalance

- Distributed key-value system



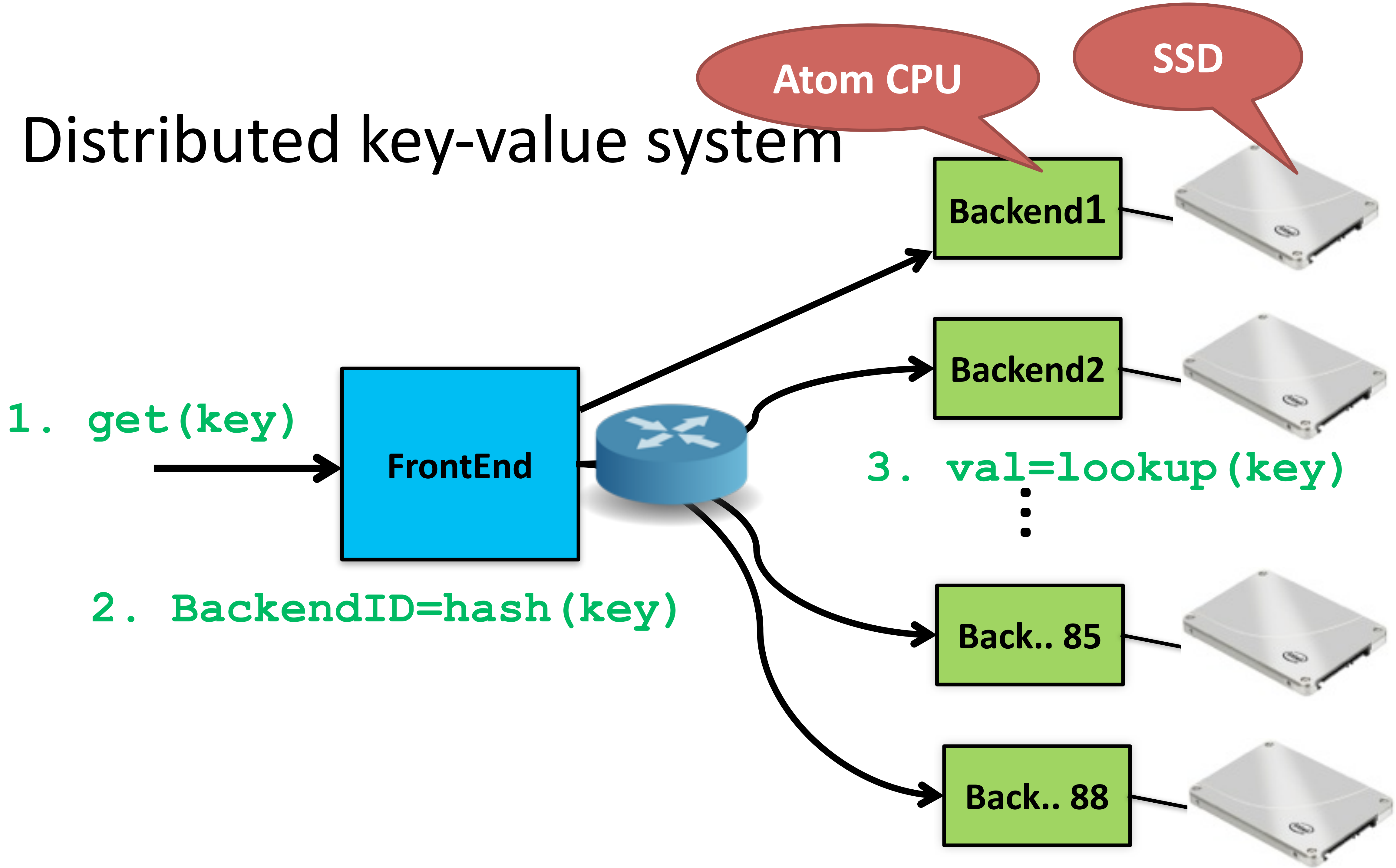
And now... Load imbalance

- Distributed key-value system



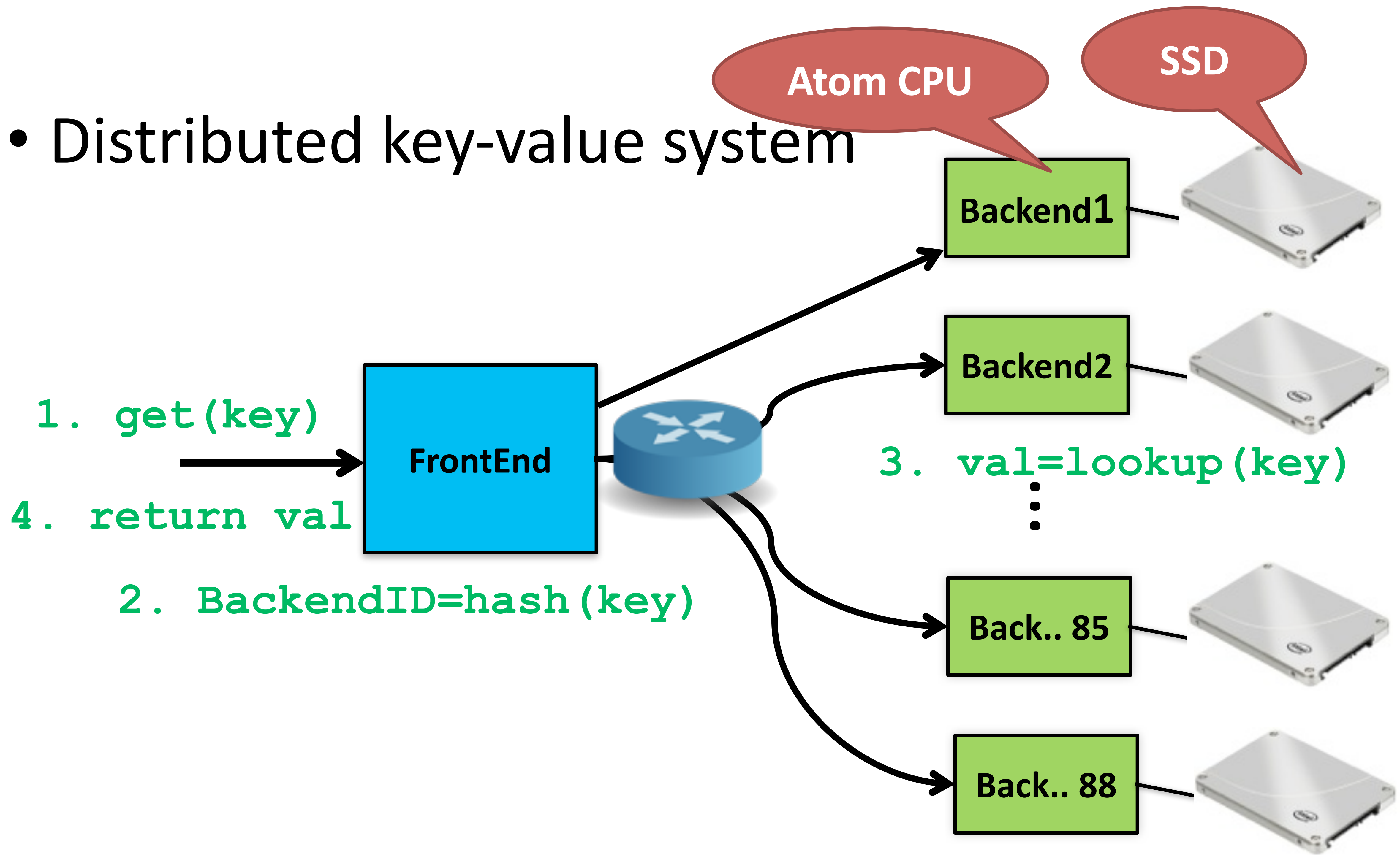
And now... Load imbalance

- Distributed key-value system



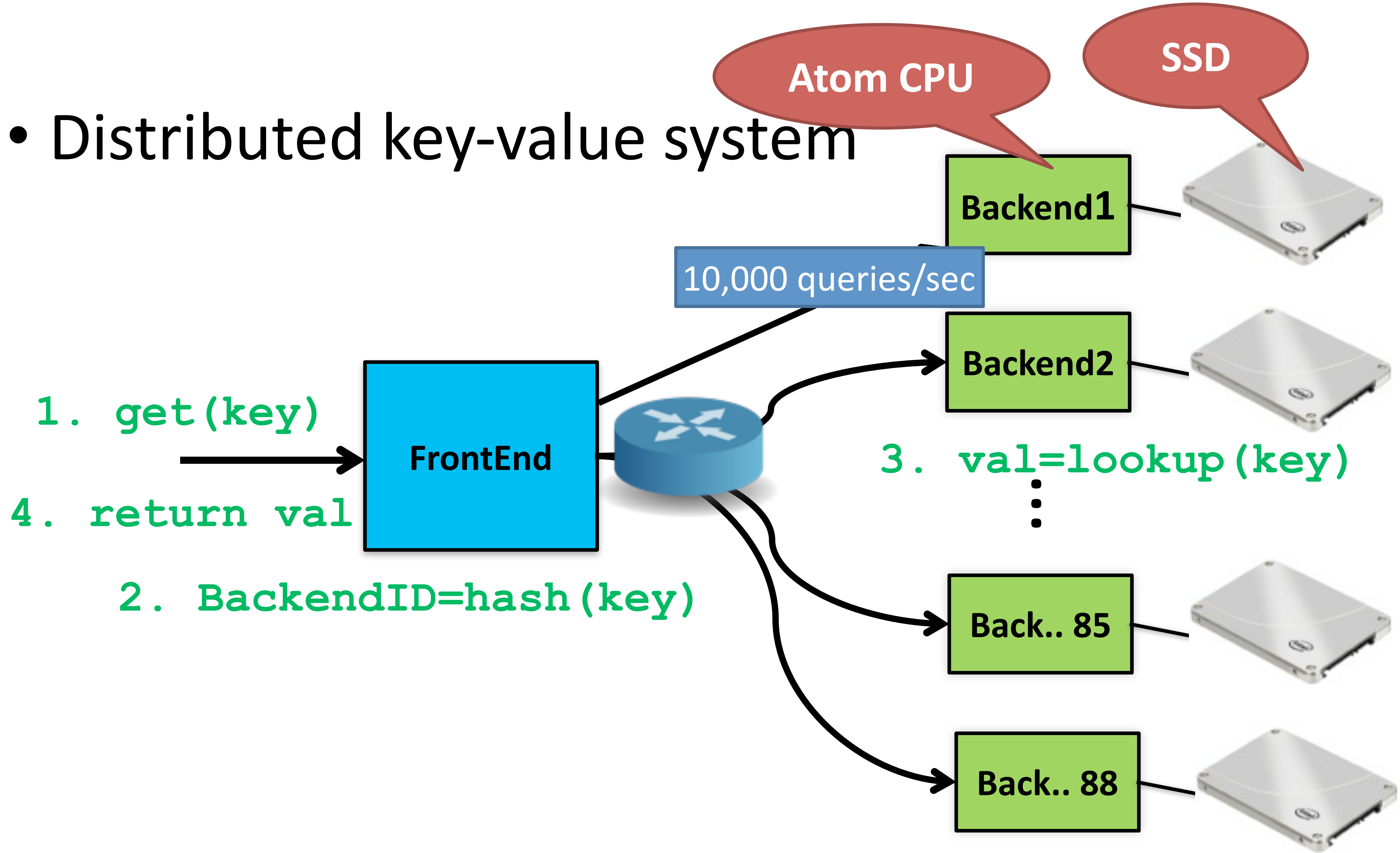
And now… Load imbalance

- Distributed key-value system



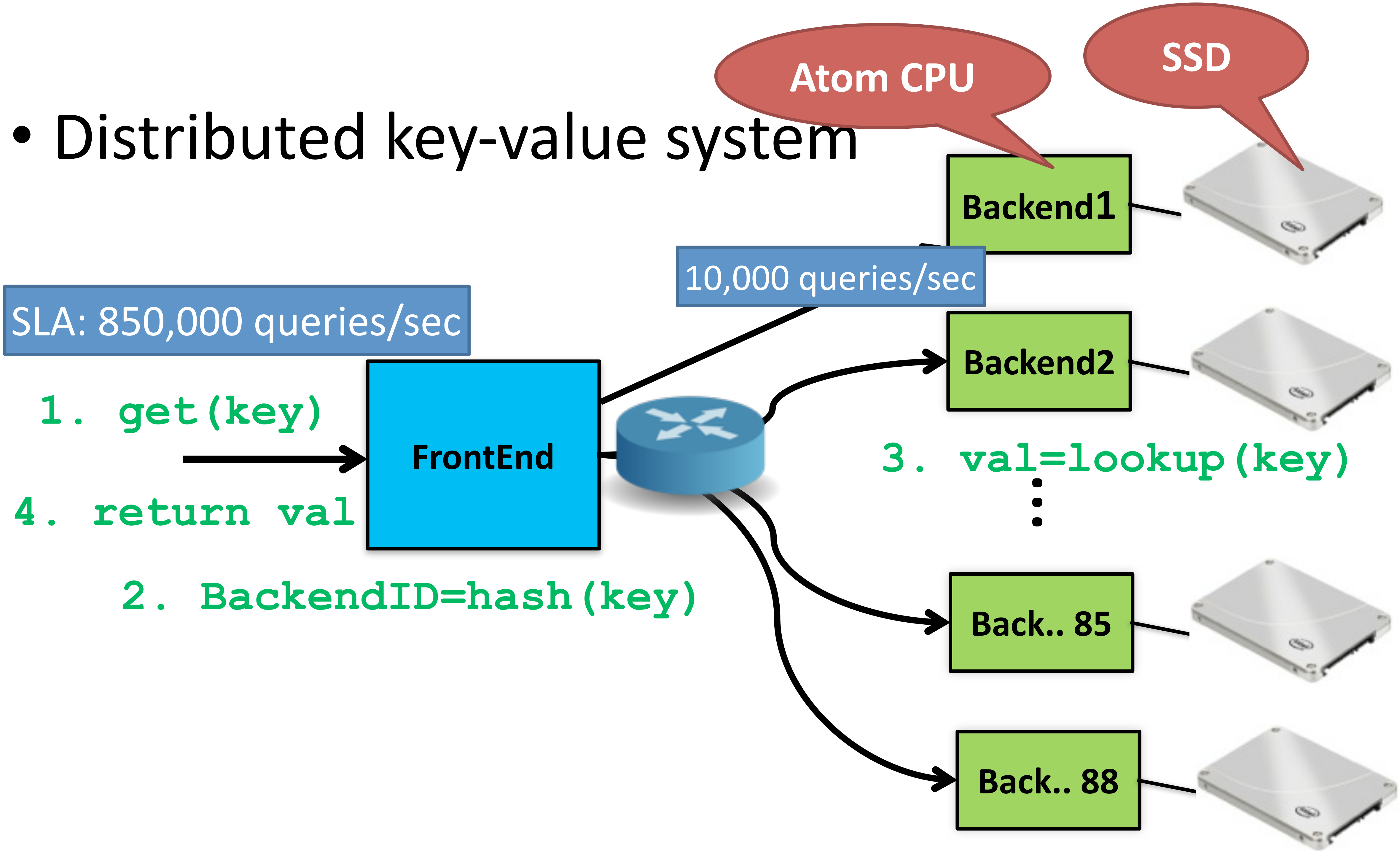
And now... Load imbalance

- Distributed key-value system

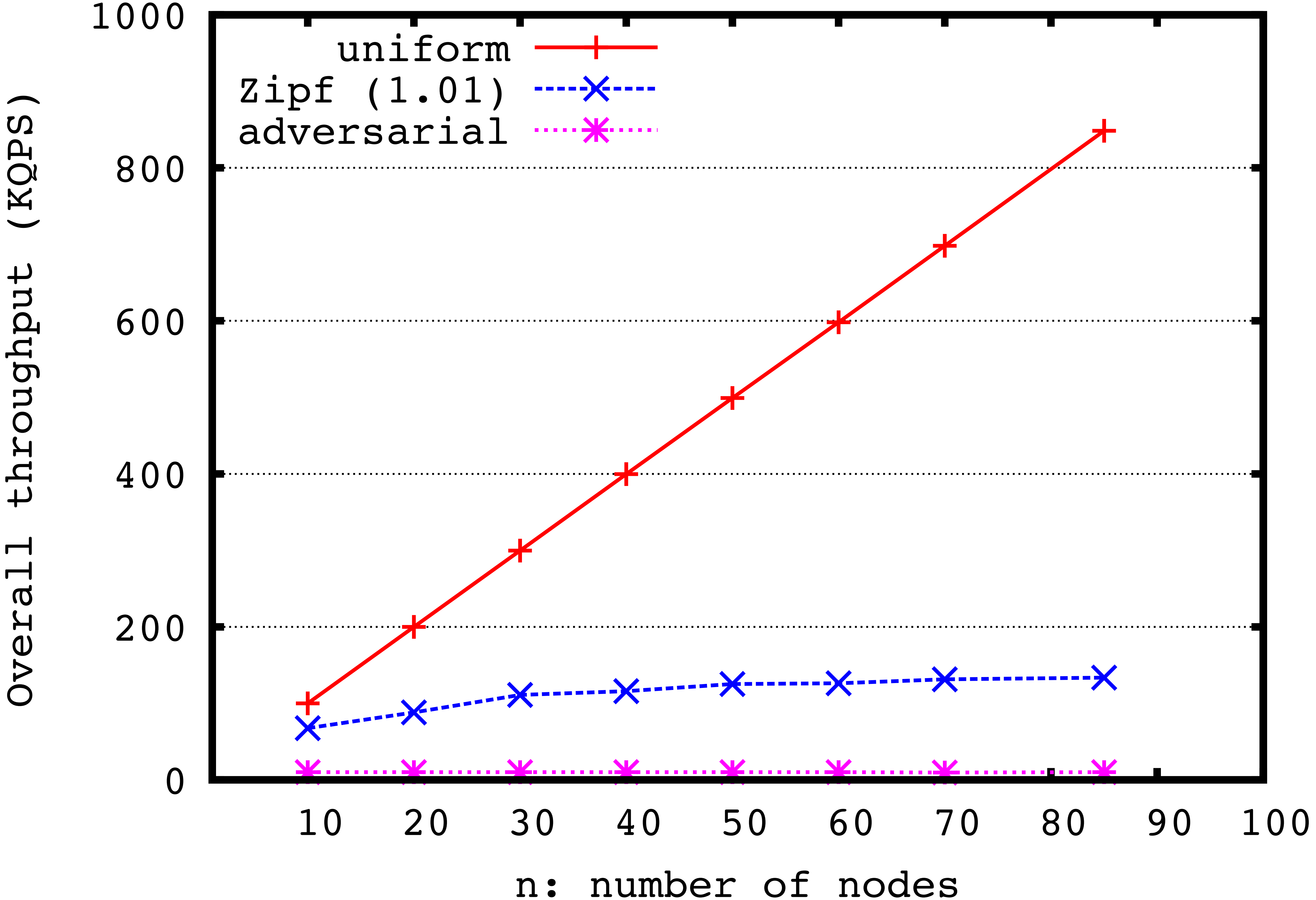


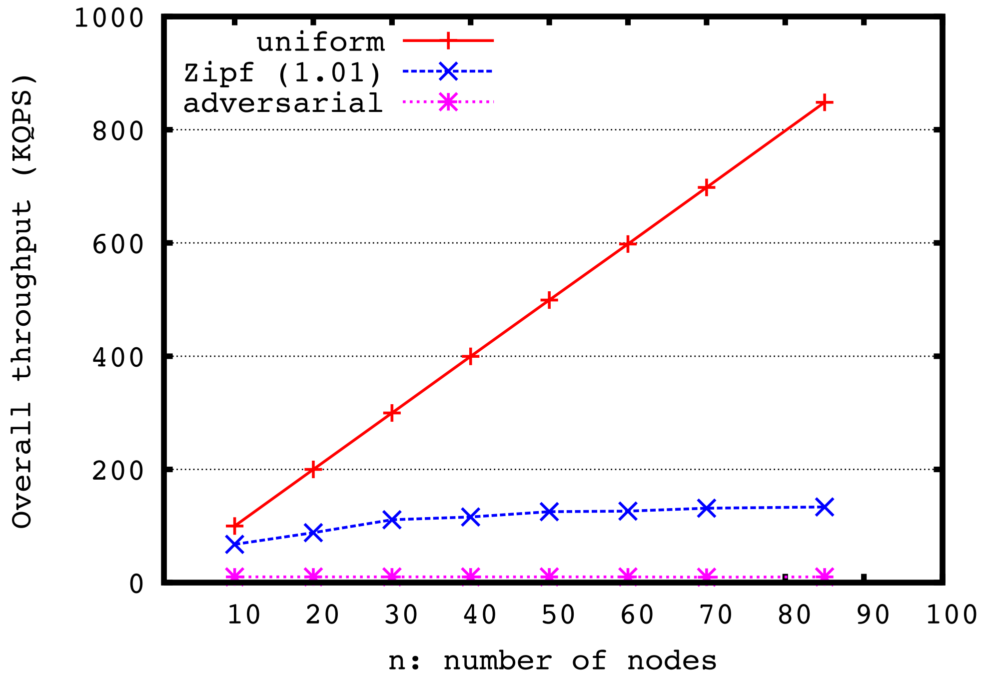
And now… Load imbalance

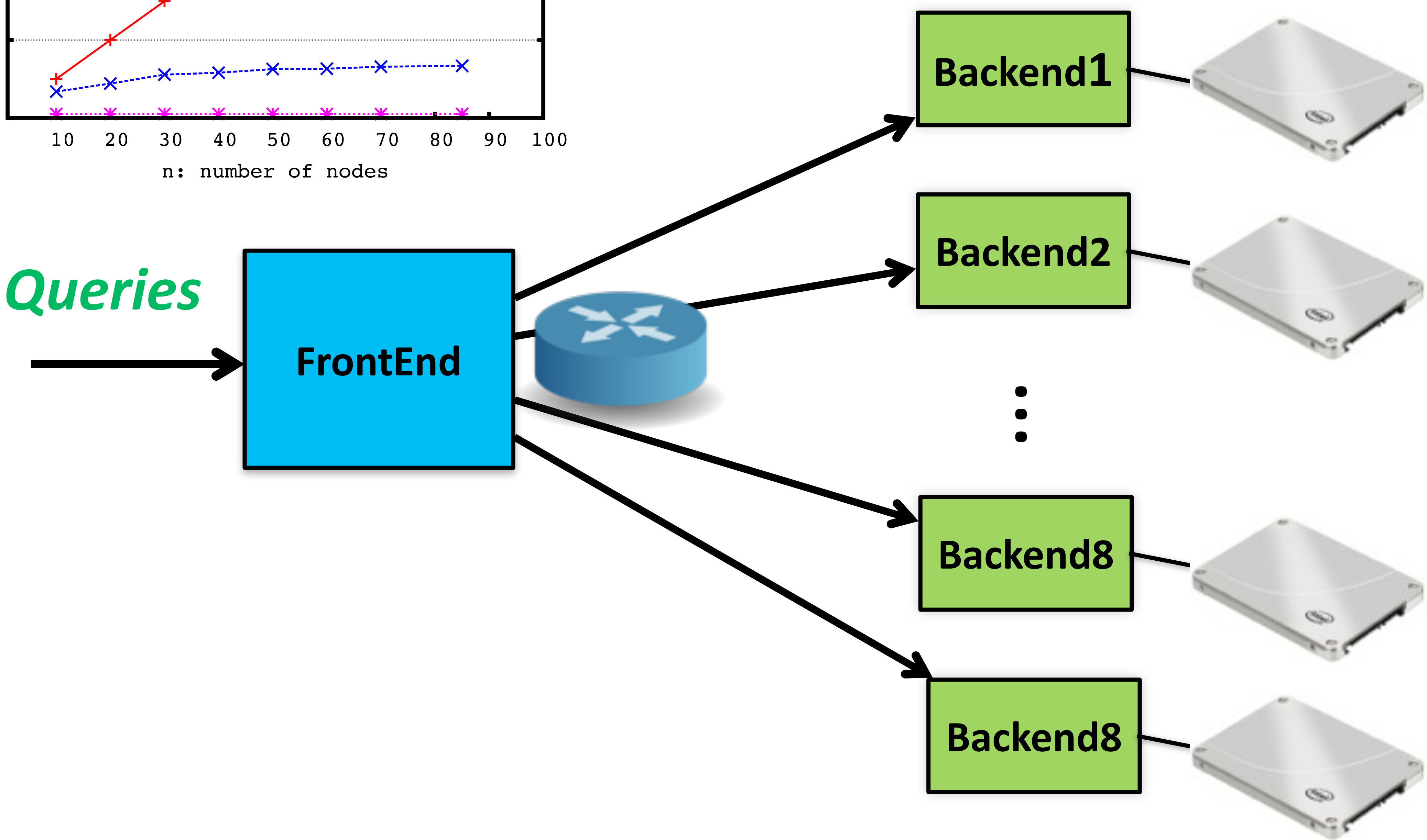
- Distributed key-value system

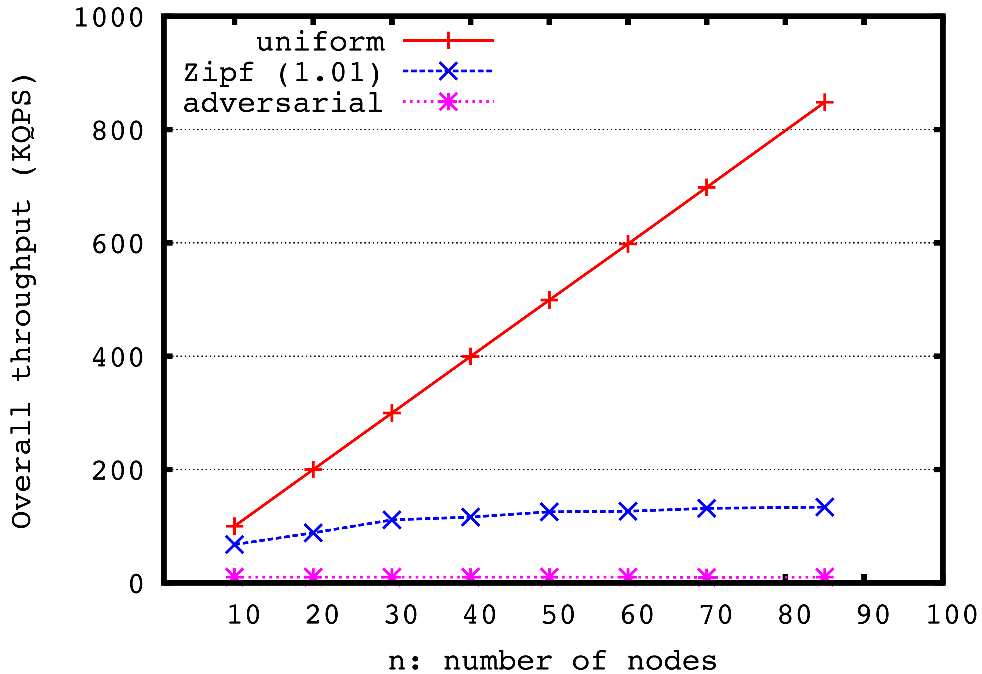


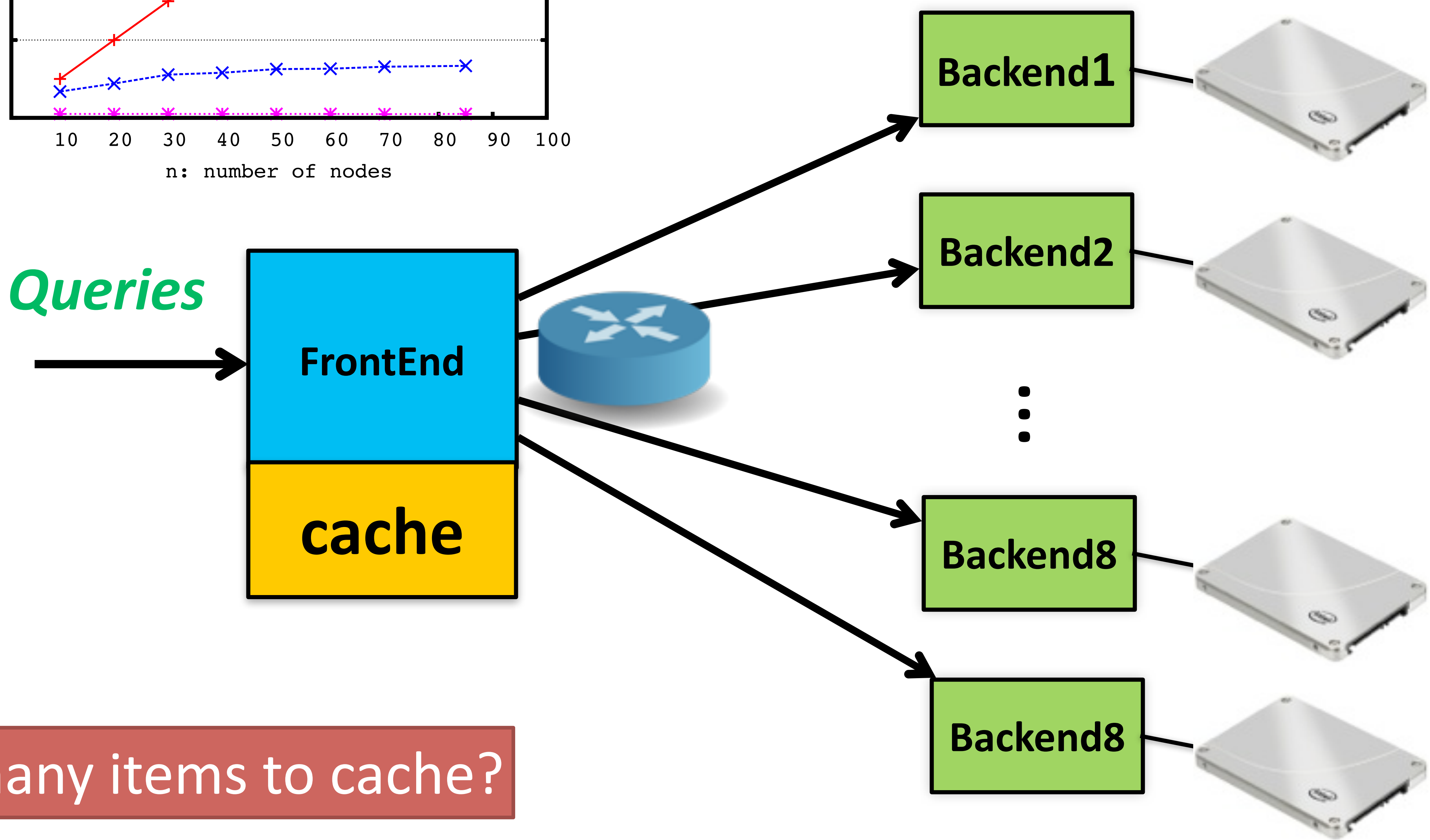
Measured tput on FAWN testbed




uniform
Zipf (1.01)
adversarial
Overall throughput (KQPS)
0
200
400
600
800
1000
10
20
30
40
50
60
70
80
90
100
n: number of nodes

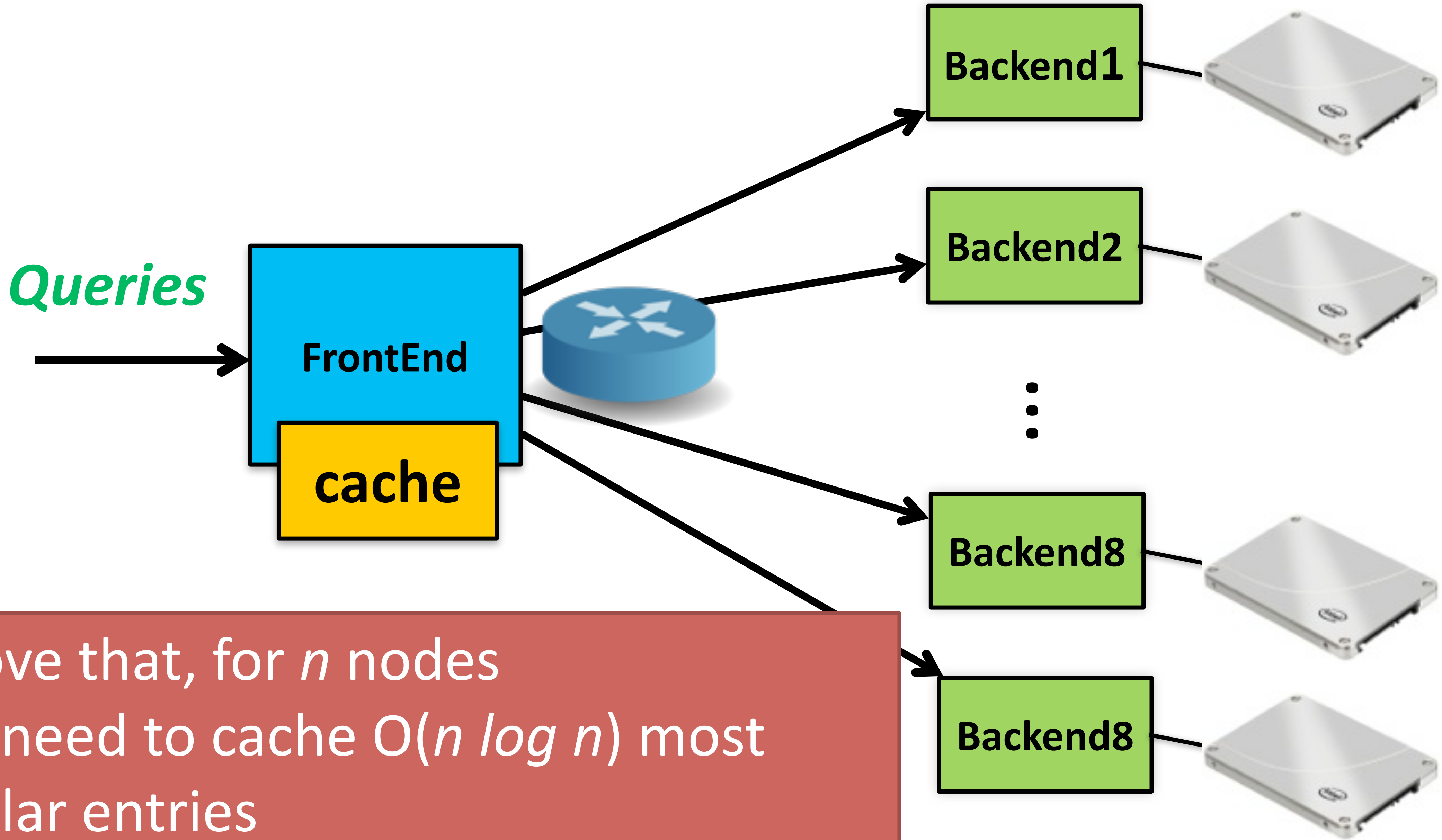

Queries
FrontEnd
Backend1
Backend2
Backend8
Backend8





How many items to cache?

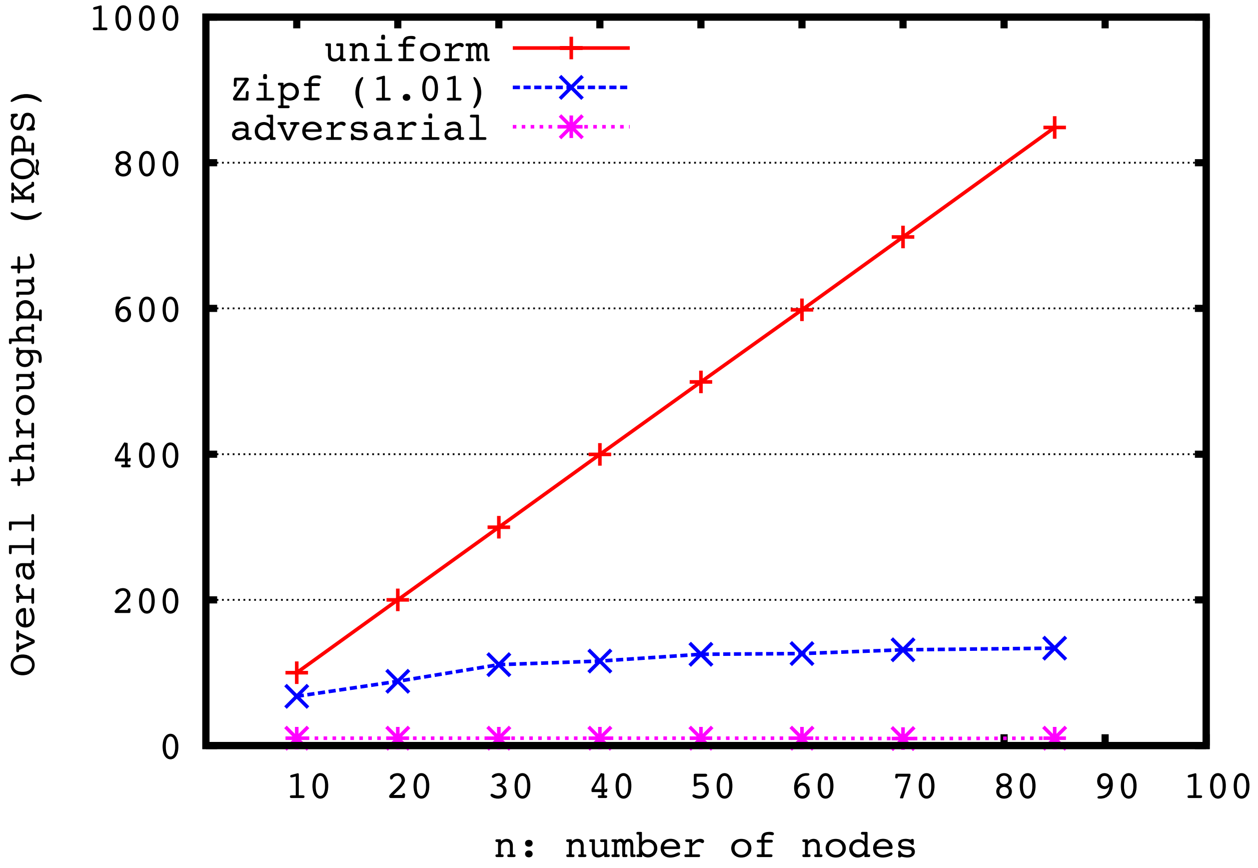
small/fast cache is enough!

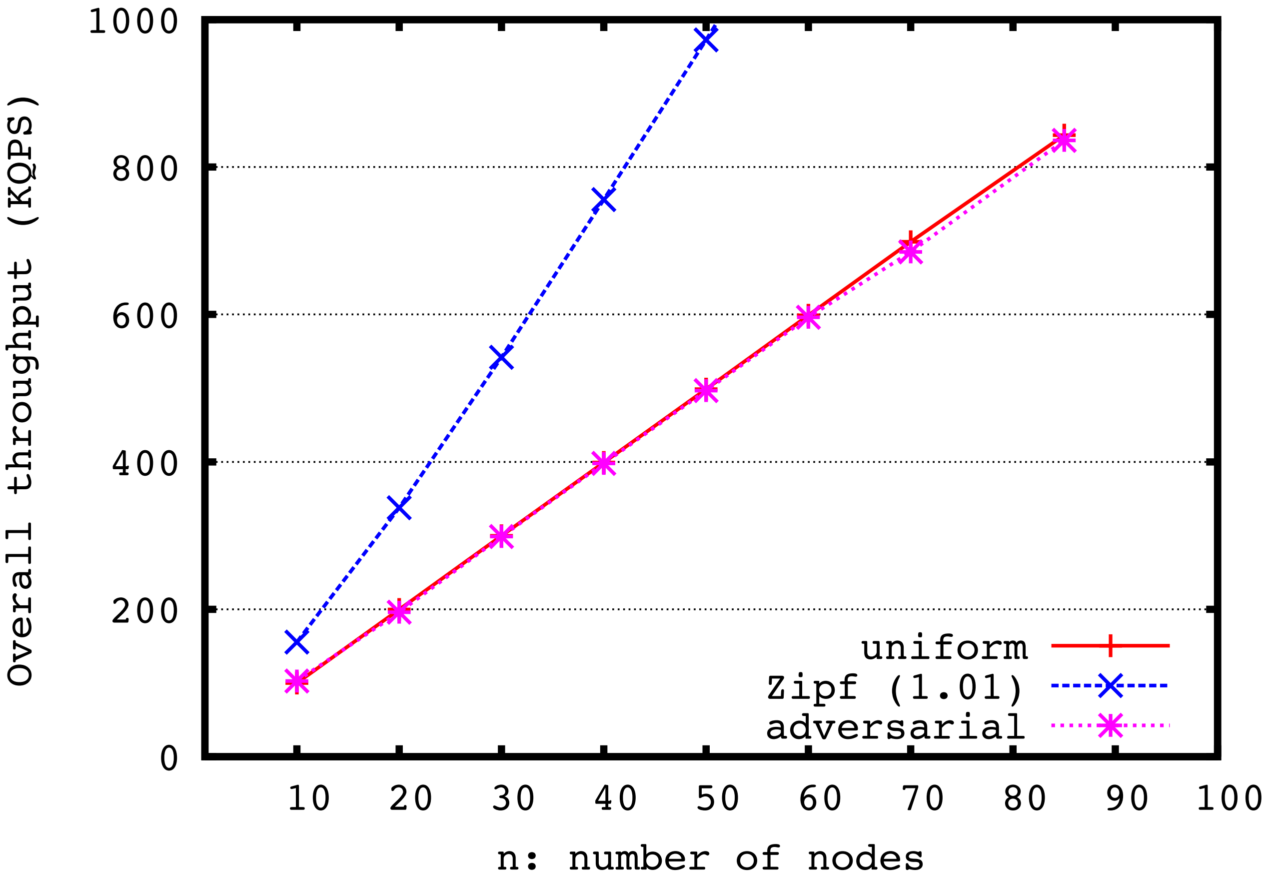


We prove that, for *n* nodes

- Only need to cache O(*n log n*) most popular entries
- With 100 backend nodes, need only about 4,000 items in the cache. Tiny!

Worst case? Now best case

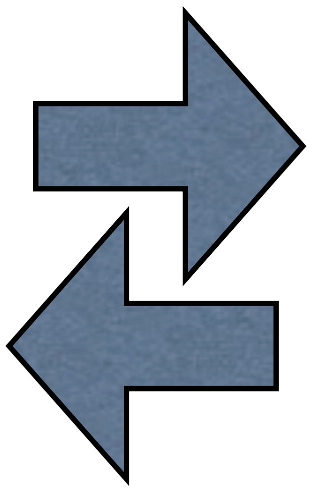




Thus...

"Wimpy" servers [FAWN, SOSP 2009]

"Brawny" server



Insanely Fast Cache

SILT

SILT

SILT

[SILT, SOSP 2011]

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

Entropy-coded tries [SOSP + ALENEX]

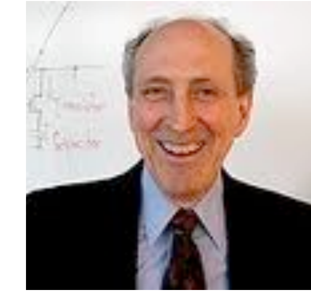
Multi-reader parallel cuckoo hashing ["MemC3" - NSDI 2013]

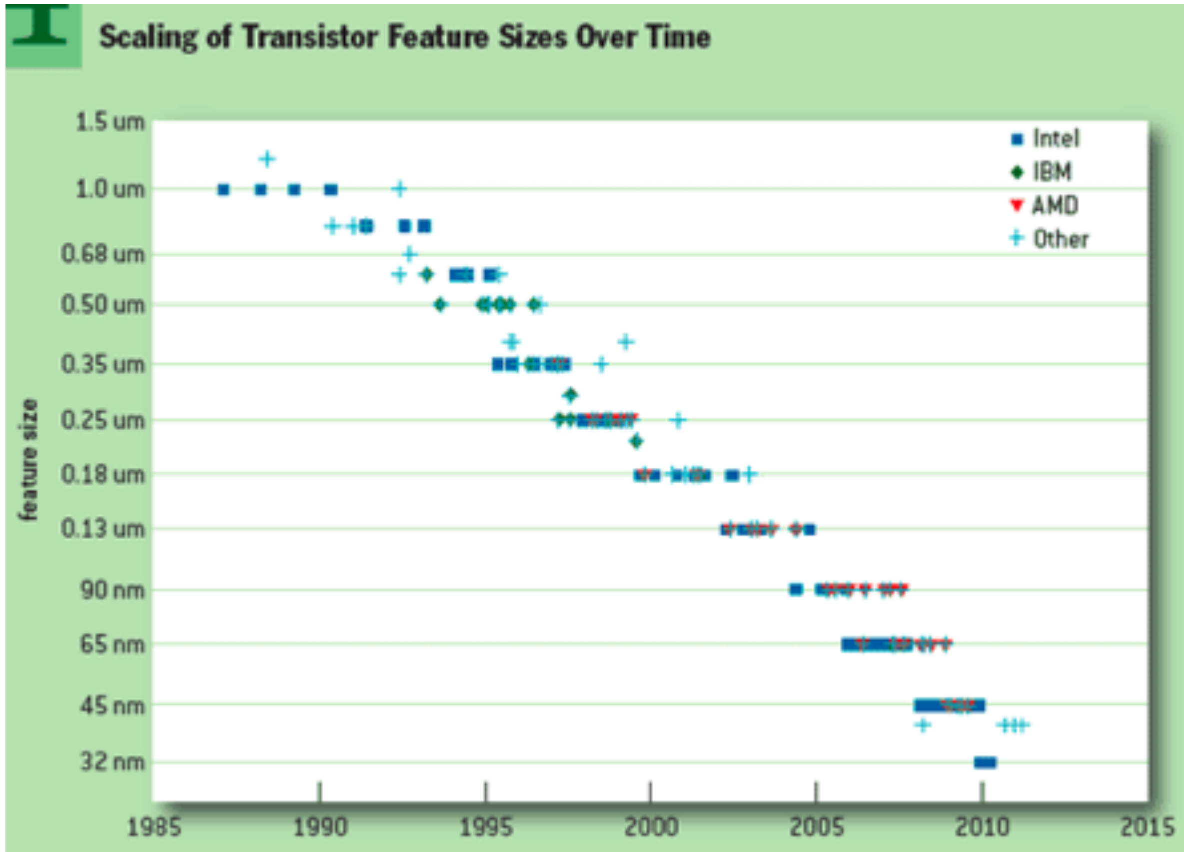
Partial-key cuckoo hashing

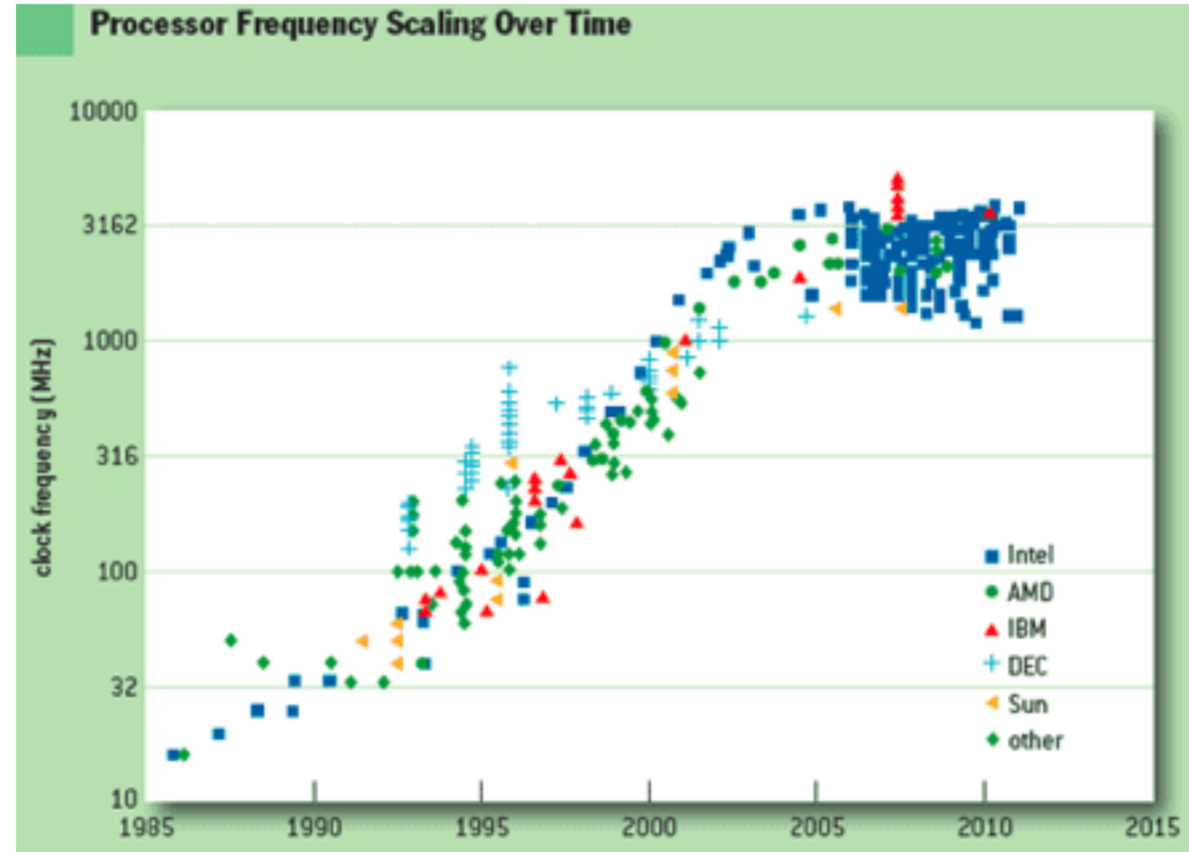
Cuckoo filter

Moore

Dennard







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

Architectures, algorithms, and programming