Ng, Xian Kai

Presents

Story

- Mr. Ball Peggert is doing **Machine Learning** research
  - Woah Machine Learning!!!
- 100petaFLOPS but ***hours*** of waiting
- Use **TIME** and **STRACE**
- **I/O** bound!?
- RAM **bandwidth** sucks

Enter

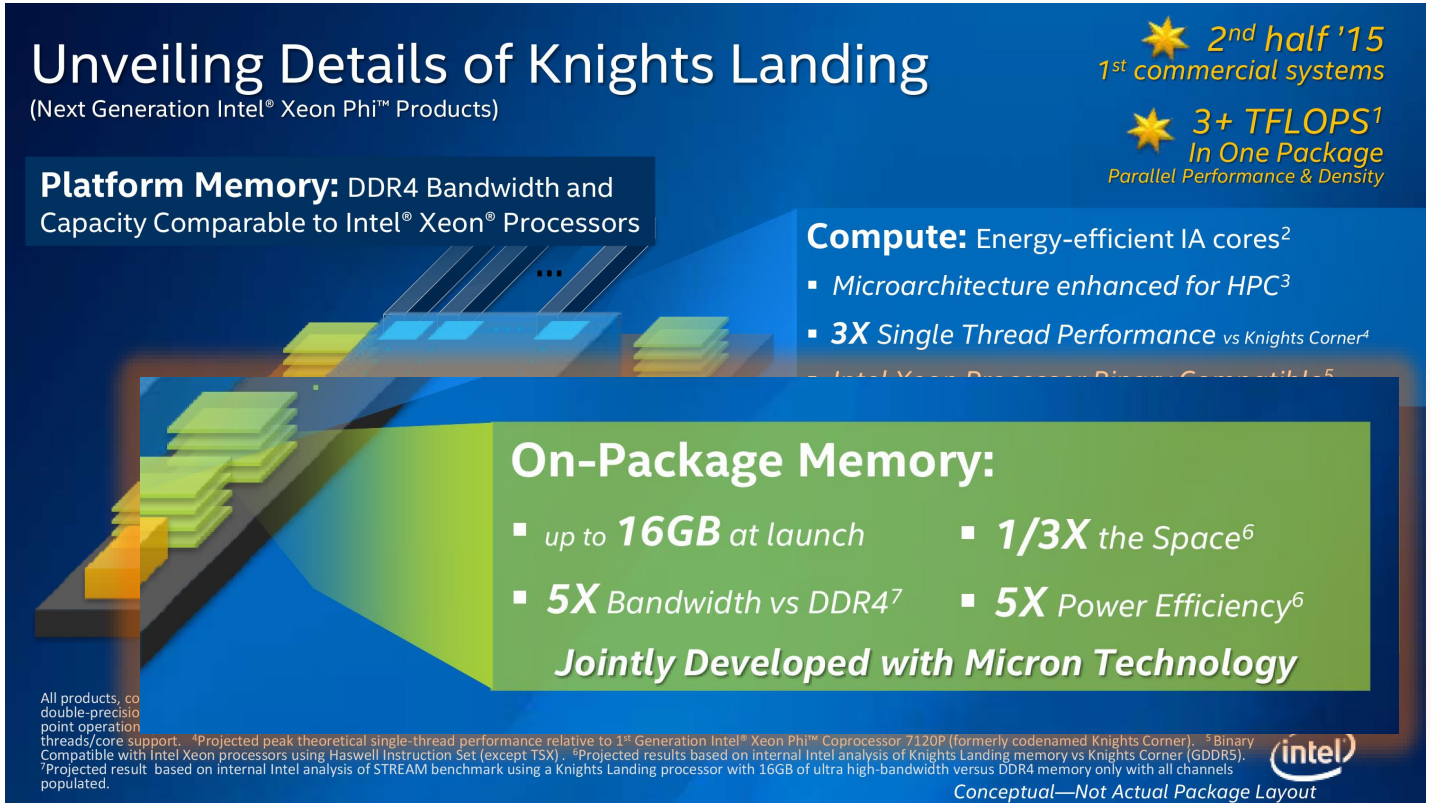
Banshee

Banshee?

Banshee:

Algorithm 1: Cache Replacement Algorithm

```
Input : tag
# rand(): random number between 0 and 1.0
if rand() < recent_miss_rate × sampling_coeff then
    meta = dram_cache.loadMetadata(tag)
    if tag in meta then
        meta[tag].count ++
        if tag in meta.candidates and meta[tag].count >
          meta.cached.minCount() + threshold then
            replace the cached page having the minimal
              counter with the accessed page
        end
        if meta[tag].count == max_count then
            # Counter overflow, divide by 2
            forall t in meta.tags do
                meta[t].count /= 2
            end
        end
        dram_cache.storeTag(tag, metadata)
    else
        victim = random page in meta.candidates
        if rand() < 1 / victim.count then
            victim.tag = tag
            victim.count = 1
            dram_cache.storeTag(tag, metadata)
        end
    end
end
```



Banshee:

Bandwidth-Efficient DRAM Caching Via Software/Hardware Cooperation

Xiangyao Yu[†], Christopher J. Hughes[‡], Nadathur Satish[‡], Onur Mutlu[§], Srinivas Devadas[†]

[†] CSAIL, MIT, [‡] Intel Labs, [§] ETH Zurich

[†]{yxy, devadas}@mit.edu, [‡]{christopher.j.hughes, nadathur.rajagopalan.satish}@intel.com,
[§]onur.mutlu@inf.ethz.ch

Presentation Goals

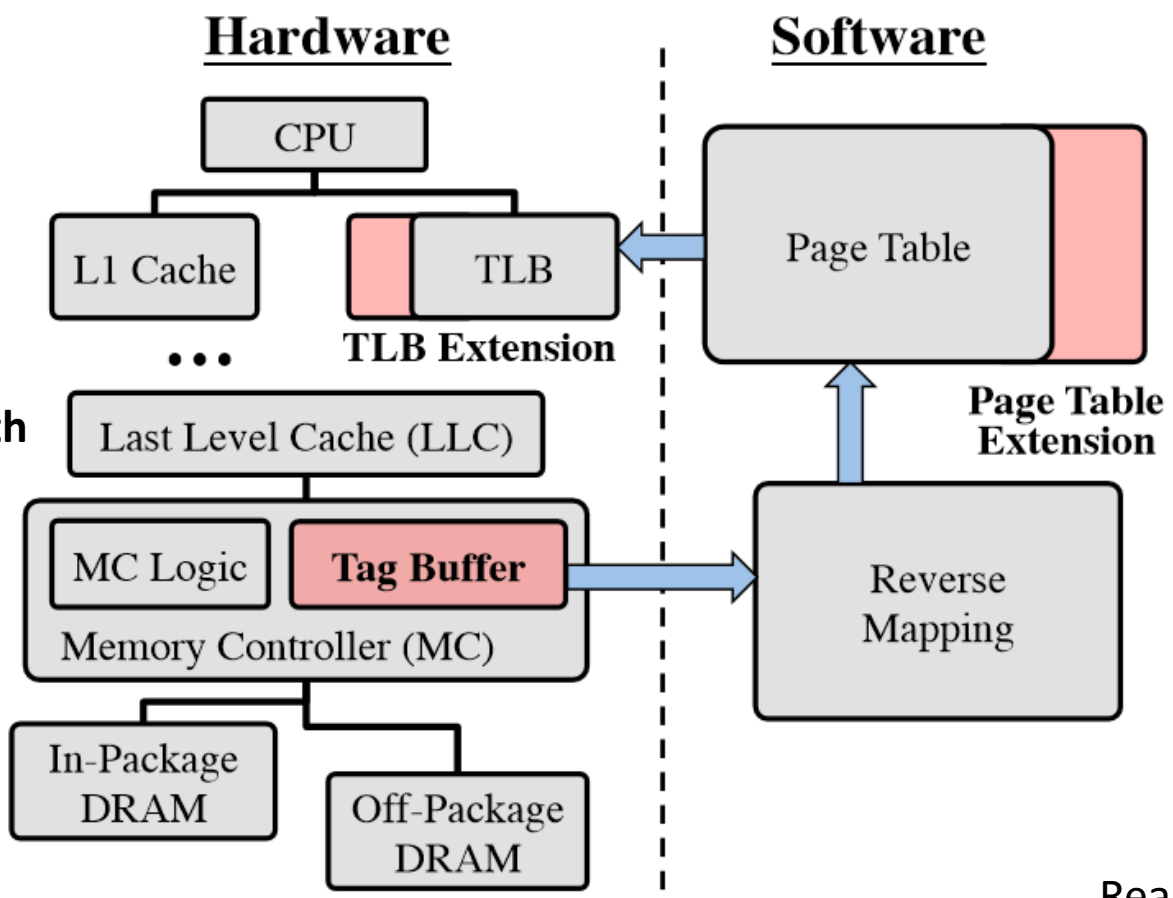
- Overview memory architecture and DRAM cache
- Explain the HAWT NEW Banshee
  - Know about new developments
    - Rest assured that my money in chip manufacturing stocks will grow
      - $.$
  - Understand mechanisms supporting ML
- Incite curiosity
  - Invite into computer architecture
    - So I can do ML and make money
- Get an A

Memory Architecture (Cache and stuff)

TLB: Virtual => Physical
Page Table: TLB's backing
(stored in main memory)

L1 to LLC cache DRAM
MC: Manages DRAM access

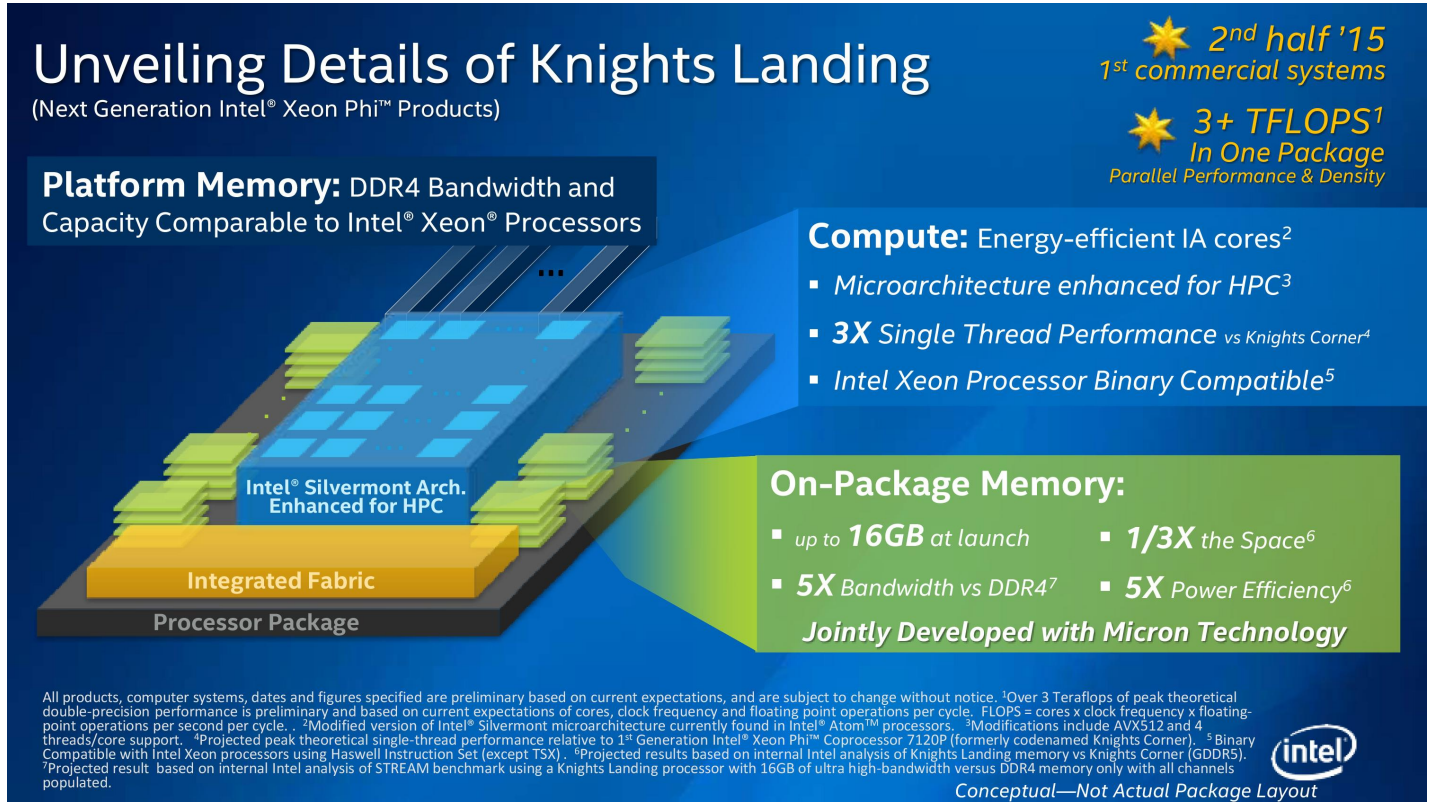
DRAM Cache: higher bandwidth



Read CS33 textbook for more

In-package DRAM

- Built together with the processor and SOC
- Higher bandwidth
- **Same latency**



Unveiling Details of Knights Landing

(Next Generation Intel® Xeon Phi™ Products)

2nd half '15
1st commercial systems

3+ TFLOPS[1]
In One Package
Parallel Performance & Density

Platform Memory: DDR4 Bandwidth and Capacity Comparable to Intel® Xeon® Processors

Compute: Energy-efficient IA cores[2]

- *Microarchitecture enhanced for HPC[3]*
- ***3X** Single Thread Performance vs Knights Corner[4]*

On-Package Memory:

- *up to **16GB** at launch*
- ***1/3X** the Space[6]*
- ***5X** Bandwidth vs DDR4[7]*
- ***5X** Power Efficiency[6]*

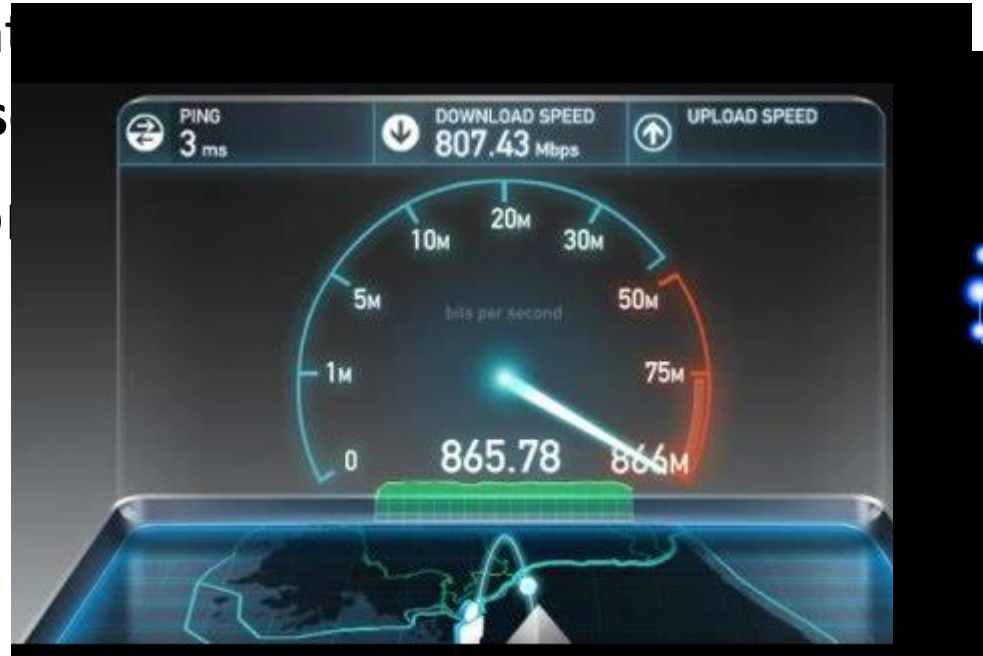
Jointly Developed with Micron Technology

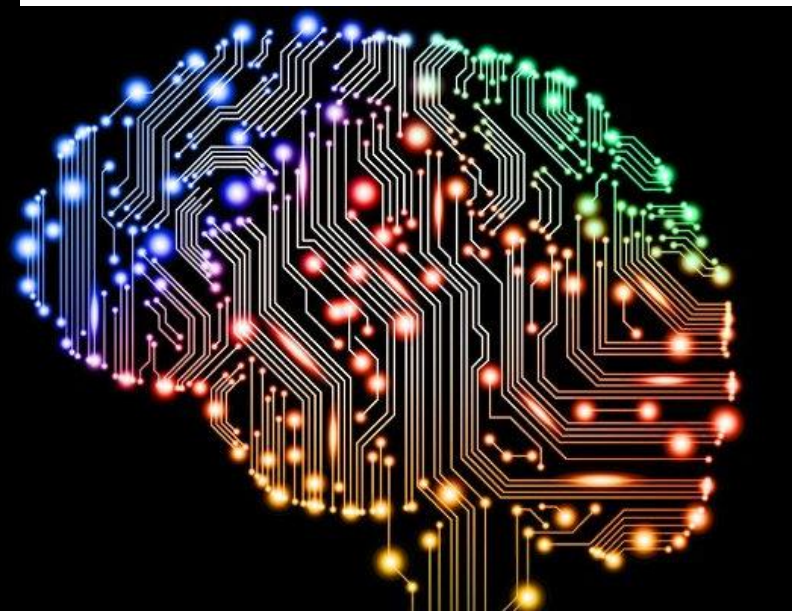
All products, compute
double-precision perf
point operations per second per cycle. . [2]Modified version of Intel® Silvermont microarchitecture currently found in Intel® Atom™ processors. [3]Modifications include AVX512 and 4 threads/core support. [4]Projected peak theoretical single-thread performance relative to 1st Generation Intel® Xeon Phi™ Coprocessor 7120P (formerly codenamed Knights Corner). [5] Binary Compatible with Intel Xeon processors using Haswell Instruction Set (except TSX) . [6]Projected results based on internal Intel analysis of Knights Landing memory vs Knights Corner (GDDR5). [7]Projected result based on internal Intel analysis of STREAM benchmark using a Knights Landing processor with 16GB of ultra high-bandwidth versus DDR4 memory only with all channels populated.

Conceptual—Not Actual Package Layout

Why Banshee

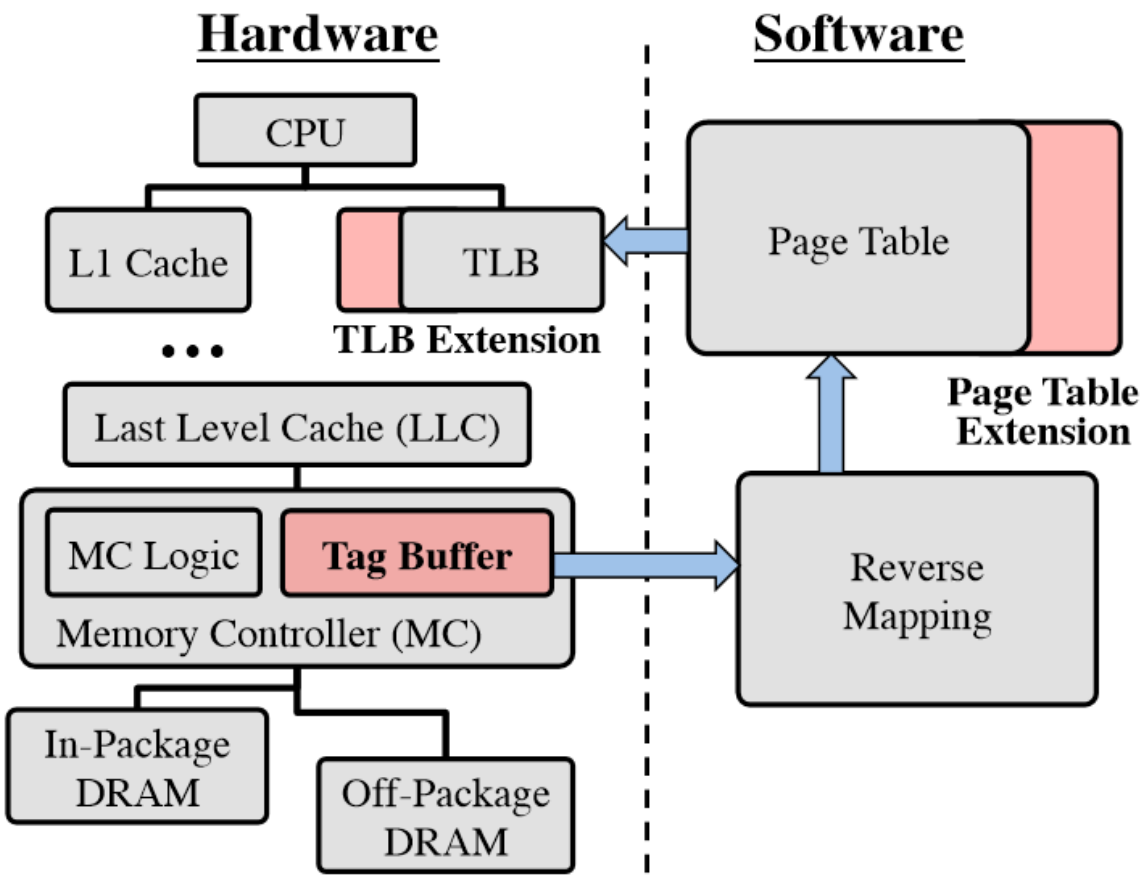
- **Bandwidth** bound modern applications
  - Graph and Machine Learning algorithms, Sparse Lin. Alg. HPC Code
- Need Caching because expensive
- Leverages ~4x bandwidth boost of **in-package** RAM
  - From Intel's Knigh
  - **Not latency-focus**
- Large Pages suppo





How Banshee

- Software and Hardware
- TLB/PTE Extension
- Tag Buffer
  - Coherence
- Sampling-based counter
  - Meta-data in DRAM cache
- Frequency-based replacement



Architecture melding Hardware **and** Software

- Just Hardware:
  - Quickly adaptive to changing program behavior
  - Dumb ‘algorithm’ (replace every time)
- Just Software:
  - Smart at predicting what data to cache
  - Memory write time overhead so runs only every so often
- Banshee:
  - Hardware tag buffer; lazy TLB coherence protocol; tracks miss rate
  - If buffer filled, consult software for how to replace

Algorithm

- Decide whether to sample
  - Check if in metadata
    - Increment count
    - If candidate, check if above threshold
    - Check overflow
  - Decide whether to add as candidate

Algorithm 1: Cache Replacement Algorithm

```
Input : tag
# rand(): random number between 0 and 1.0
if rand() < recent_miss_rate × sampling_coeff then
    meta = dram_cache.loadMetadata(tag)
    if tag in meta then
        meta[tag].count ++
        if tag in meta.candidates and meta[tag].count >
          meta.cached.minCount() + threshold then
            replace the cached page having the minimal
              counter with the accessed page
        end
        if meta[tag].count == max_count then
            # Counter overflow, divide by 2
            forall t in meta.tags do
                meta[t].count /= 2
            end
        end
        dram_cache.storeTag(tag, metadata)
    else
        victim = random page in meta.candidates
        if rand() < 1 / victim.count then
            victim.tag = tag
            victim.count = 1
            dram_cache.storeTag(tag, metadata)
        end
    end
end
```

DRAM Cache

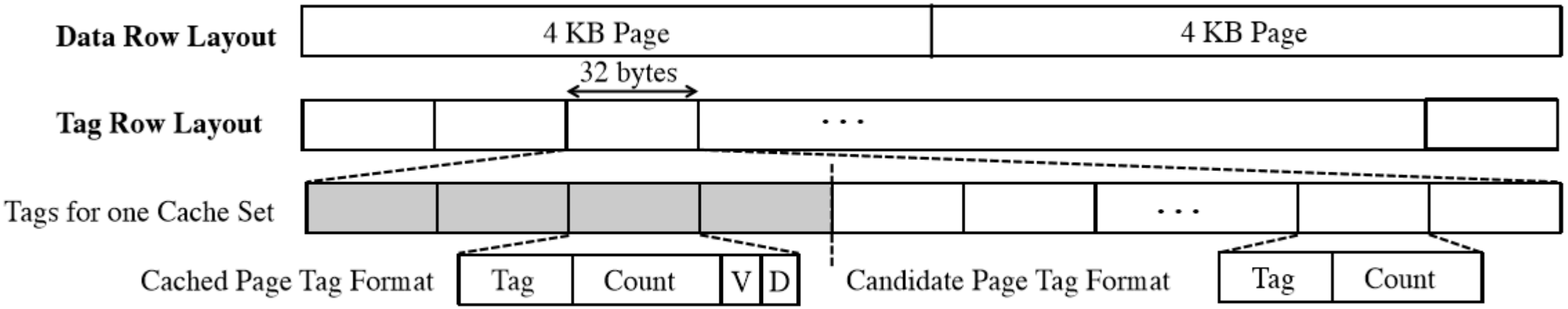


Figure 3: 4-way associative DRAM cache layout (not drawn to scale).

PTE and TLB Extension bits

- Physical address are same for in and off package RAM
  - Address consistency issue => on-chip cache scrubbing
- Cached bit: In-package?
- Way bit: Where in package?
- Overhead: 4-way associative = 2 way bits + 1 cached bit = 3 bits
  - 4% for TLB
  - 0% for PTE (since it has unused bits)

Tag buffer

- Physical address as tag
- Cached bit
- Way bit
- Valid bit
- Remap bit

Speed

- Over 3x faster with page rank program than no cache
- Faster than only in-package RAM (LOL?)
  - Because both in and off package = more throughput
- VS. Competing algorithms/architecture
  - 68.9% speedup over Unison Cache
  - 26.1% over TDC
  - 15.0% over Alloy Cache

Links

- MIT News article: http://news.mit.edu/2017/new-high-capacity-data-caches-more-efficient-1023
- Paper: https://arxiv.org/abs/1704.02677
- Pics: AncientPages.com, Intel, speedtest.net, LinkedIn