ECE154A — Discussion 10

George Higgins Hutchinson

December 3, 2021

Keep your eyes open for...

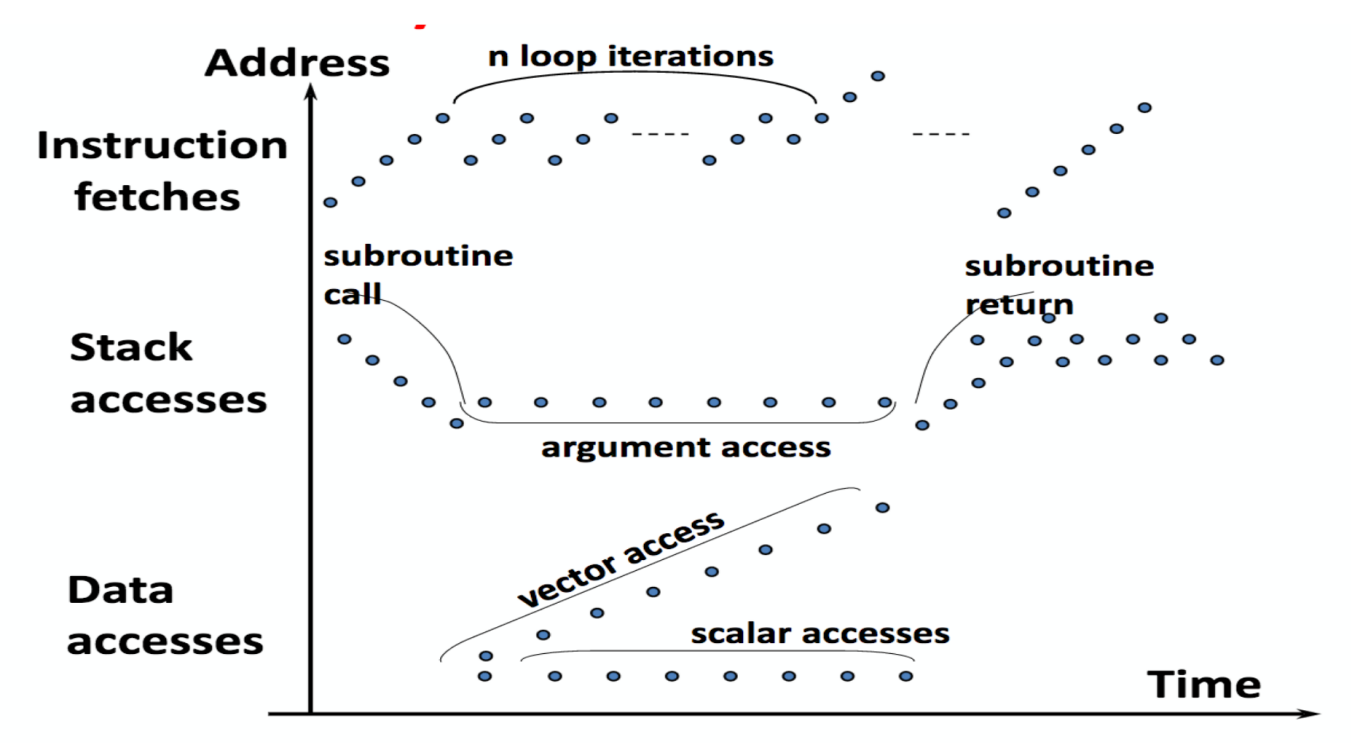
- Lab 6: due Dec 3
- HW6: due Dec 3
- Review OH: TBA+poll

Caches

Why cache? Locality

Memory access in programs follows some useful patterns.

Locality in space (access nearby locations) and time (access the same location in quick succession).



Why cache?

- Exploit temporal locality by pulling data into faster memory
- Exploit spatial locality by pulling more data around any access (block size)
- Cache INPUT: a memory address
- Cache OUTPUT: data in memory at that location
- Access time: now stochastic (overall pipeline needs to be designed around this)

Addressing a cache

Break down an address into Tag, Index, and Offset:

Tag (leftover bits)	Index (which row)	Offset (which byte within the block)

Tag bits: need to store to check that this is actually what you're accessing

- # of bits in address - # index bits - # offset bits

Index bits:

- lg(cache size / block size / associativity)

Offset bits:

- lg(block size)

Sizing a Cache

- Total bits for a cache block: offset + data + dirty (1bit) + valid (1bit)
- ‘is general, you may see special designs that need more/different metadata
- Valid bit: used to avoid giving garbage data on startup (more complicated if multi-core)
- Dirty bit: is the data in cache same as data in main memory?

Cache organization

- Direct map vs. Set Associative vs. Fully Associative
- Match to a set by index, need to check tags accross an entire set
- Convenient visualization: a grid with sets as rows
- Direct map $=$ 1 way associative
- More associativity $=$ shorter index, longer tag (slower comparison)

Cache Policy

- Write-back vs write-through — do you cache stores as well as loads?
- write-allocate — do you bring data into cache on a missed store?
- Commonly seen: write-through, no write-allocate, or write-back and write-allocate

Taxonomy of Cache Misses

- Compulsory: we've never seen this data before
- Capacity: wouldn't have happened if infinitely large cache
- Conflict: could have been mitigated with a same-size, more associative cache (usually means that there's a memory access pattern that wastes entire sets)

Cache practice

Consider an 8-block direct mapped cache, block size = 1 word

```
#define LARGE 16 //2 times total cache blocks
int a[LARGE];
int sum = 0
for (int z = 0; z < 2; z++)
  for (int x = 0; x < LARGE; x++)
    sum += a[x];

//other stuff (assume flushed cache)

sum = 0;
for (int z=0; z<2; z++)
  for (int x=0; x<LARGE; x+=8)
    sum += a[x]
```

What misses on first time through loop? Second?

Cache practice

Consider an 8-block direct mapped cache, block size = 1 word

```
#define LARGE 16 //2 times total cache blocks
int a[LARGE];
int sum = 0
for (int z = 0; z < 2; z++)
  for (int x = 0; x < LARGE; x++)
    sum += a[x];

//other stuff (assume flushed cache)

sum = 0;
for (int z=0; z<2; z++)
  for (int x=0; x<LARGE; x+=8)
    sum += a[x]
```

What misses on first time through loop? Second?

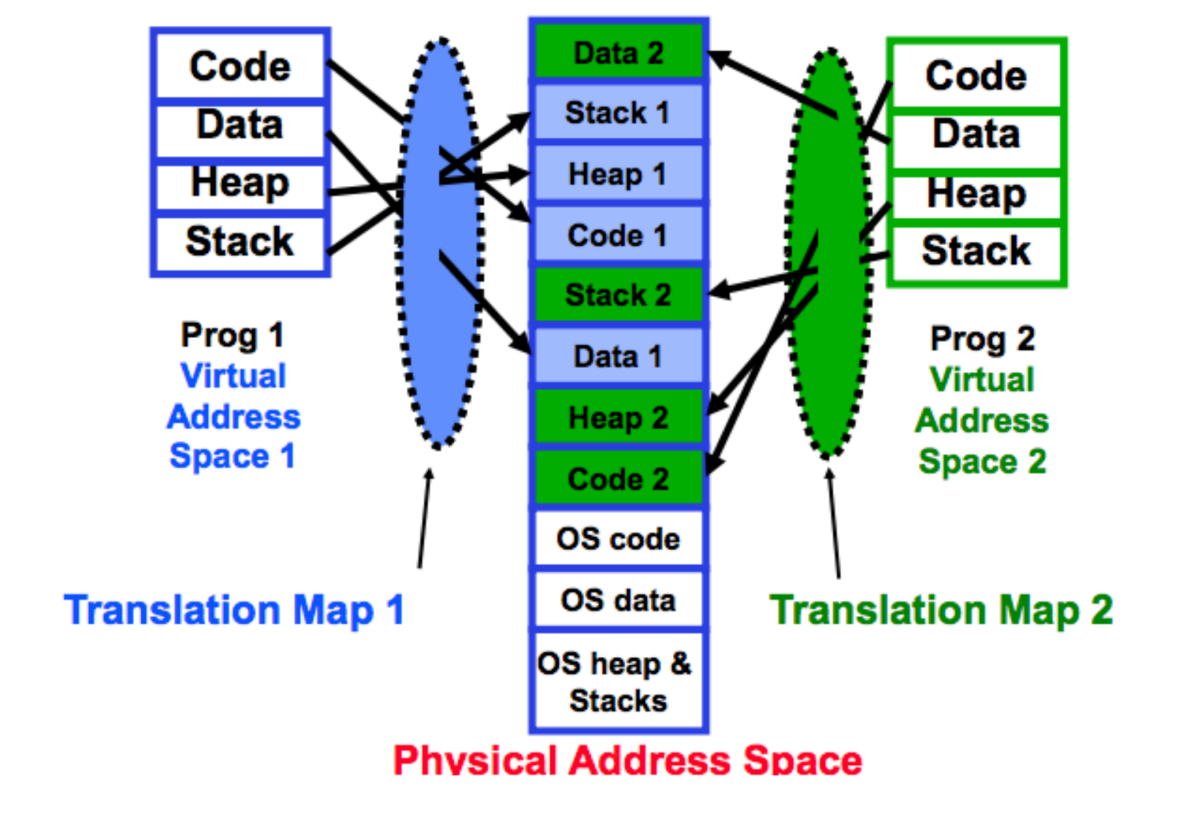
z=0: all compulsory misses. First z=1, capacity misses. Second z=1, conflict.

Virtual Memory

Why virtual memory?

- Provide programs a consistent, isolated memory space
- Decouple physical memory size from what programs see
- Isolate programs from OS and each other

Virtual Memory in Action



TLB: a cache for the page table

- With naive paging, takes at least two physical memory access to perform one effective access
- Page tables are often hierarchical to save space: penalty gets even worse
- TLBs are *often* (but don't need to be) fully-associative and small enough to be accessed in one cycle

TLB practice (adapted from final:fa'20)

A paging system has the following properties:

- TLB is fully-associative with 4 entries, each carrying a PPN
- Virtual address space is 32-bit
- Physical address space is 24-bit
- Page size: 32 KiB

How many bits in the virtual address regions? physical? how many total bits in a TLB entry? how many stored to correctly do LRU replacement?

TLB Practice Answers

Virtual address:

17-bit VPN	15-bit Page Offset

Physical address:

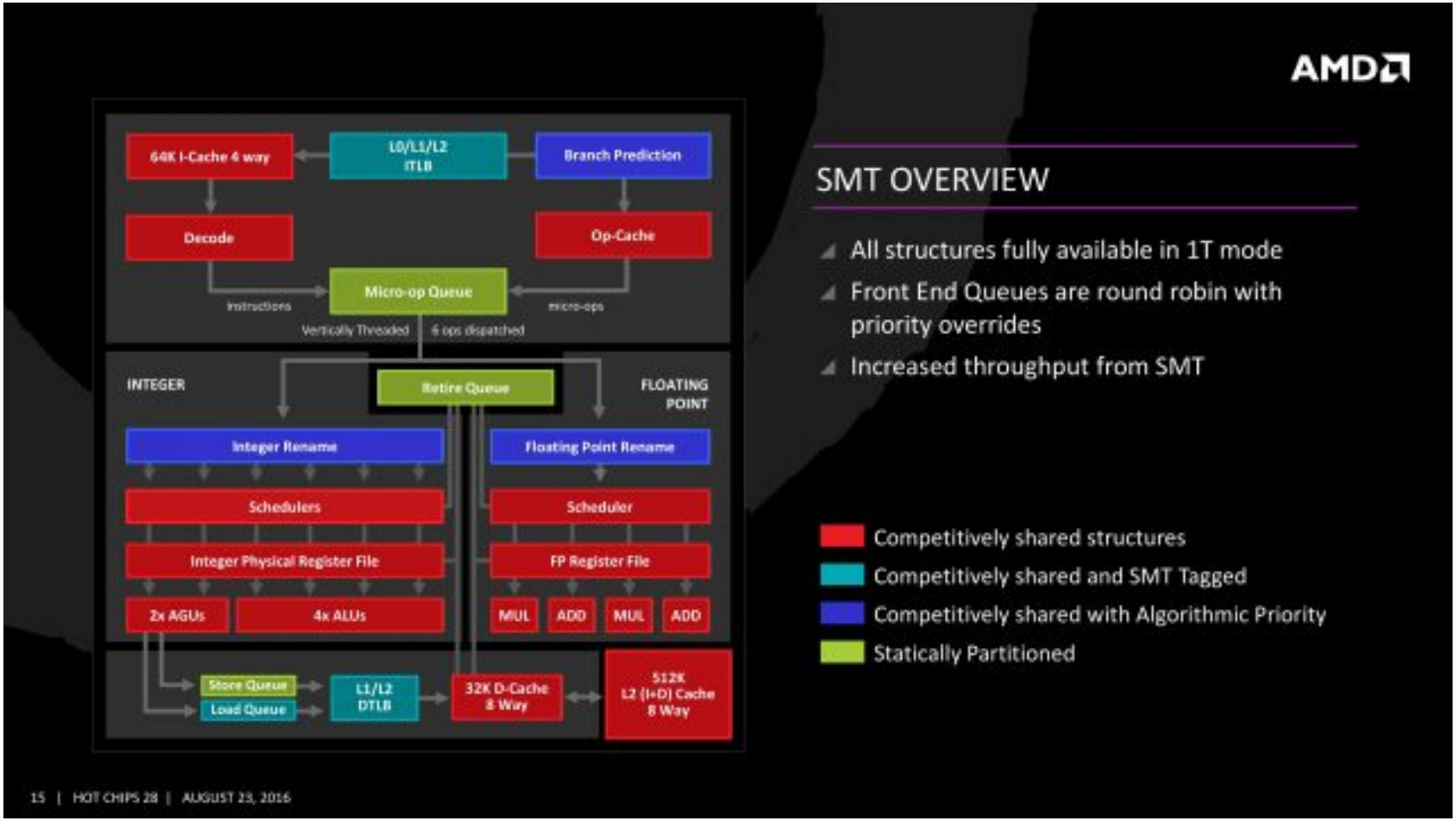
9-bit PPN	15-bit Page Offset

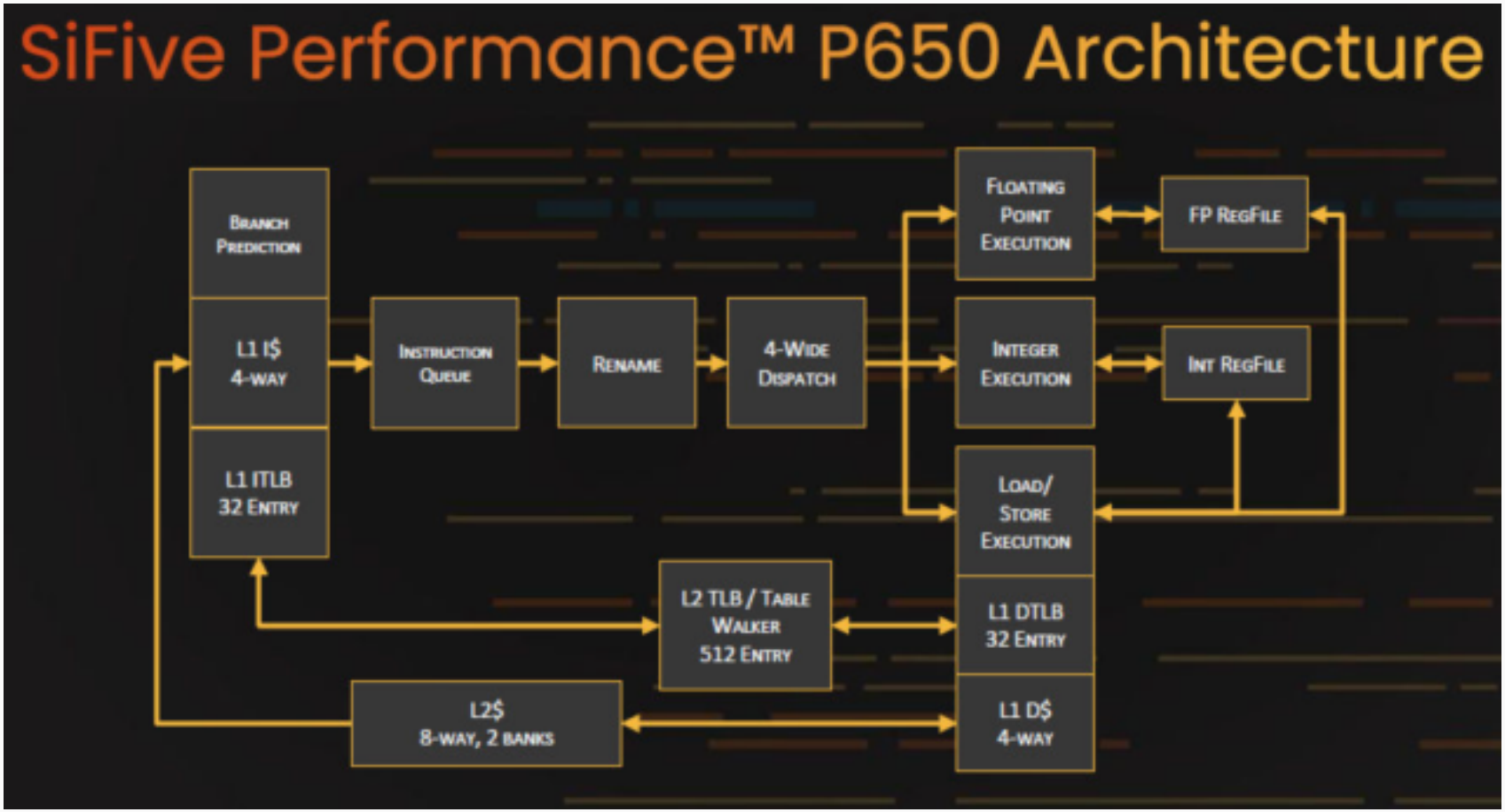
Cache entry: 17-bit tag (full VPN) + 1 valid + 9-bit PPN

LRU state: $4! = 24$ possible use histories for 4 entries, need 5 bits

That's all! AMA with any time left!

But there's more to learn about Processor Architecture




SiFive Performance™ P650 Architecture
Branch Prediction
L1 I$
4-way
L1 ITLB
32 Entry
Instruction Queue
Rename
4-Wide Dispatch
Floating Point Execution
Integer Execution
Load/ Store Execution
FP RegFile
Int RegFile
L2 TLB / Table Walker
512 Entry
L1 DTLB
32 Entry
L2$
8-way, 2 banks
L1 D$
4-way