

Lecture 16: Cache Innovations / Case Studies

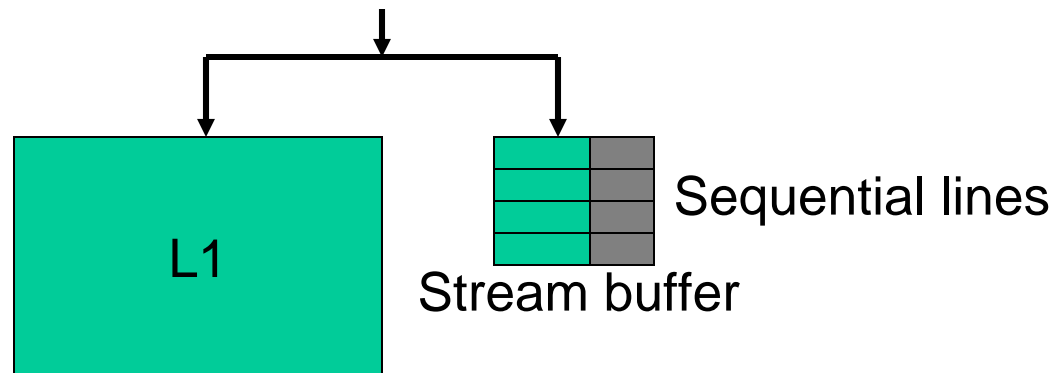
- Topics: prefetching, blocking, processor case studies (Section 5.2)

Prefetching

- Hardware prefetching can be employed for any of the cache levels
- It can introduce cache pollution – prefetched data is often placed in a separate prefetch buffer to avoid pollution – this buffer must be looked up in parallel with the cache access
- Aggressive prefetching increases “coverage”, but leads to a reduction in “accuracy” → wasted memory bandwidth
- Prefetches must be timely: they must be issued sufficiently in advance to hide the latency, but not too early (to avoid pollution and eviction before use)

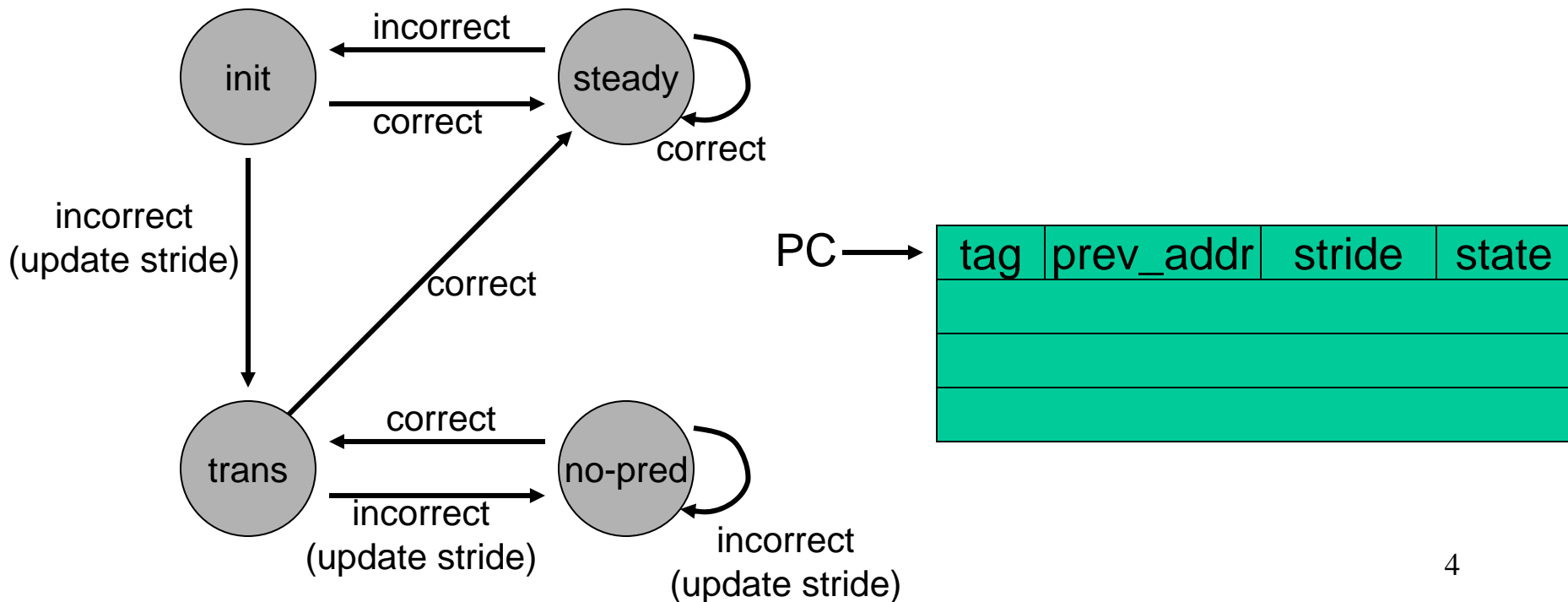
Stream Buffers

- Simplest form of prefetch: on every miss, bring in multiple cache lines
- When you read the top of the queue, bring in the next line



Stride-Based Prefetching

- For each load, keep track of the last address accessed by the load and a possibly consistent stride
- FSM detects consistent stride and issues prefetches



Compiler Optimizations

- Loop interchange: loops can be re-ordered to exploit spatial locality

```
for (j=0; j<100; j++)  
  for (i=0; i<5000; i++)  
    x[i][j] = 2 * x[i][j];
```

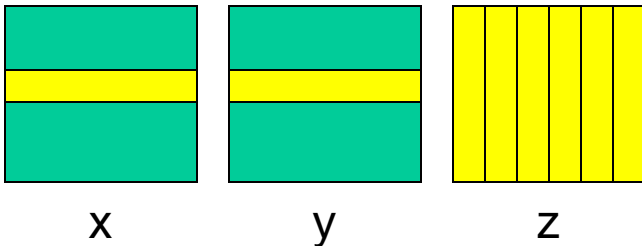
is converted to...

```
for (i=0; i<5000; i++)  
  for (j=0; j<100; j++)  
    x[i][j] = 2 * x[i][j];
```

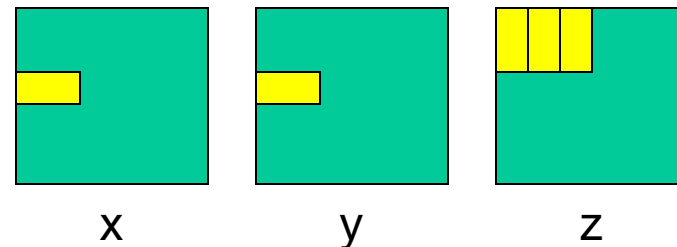
Exercise

- Re-organize data accesses so that a piece of data is used a number of times before moving on... in other words, artificially create temporal locality

```
for (i=0;i<N;i++)  
  for (j=0;j<N;j++) {  
    r=0;  
    for (k=0;k<N;k++)  
      r = r + y[i][k] * z[k][j];  
    x[i][j] = r;  
  }
```

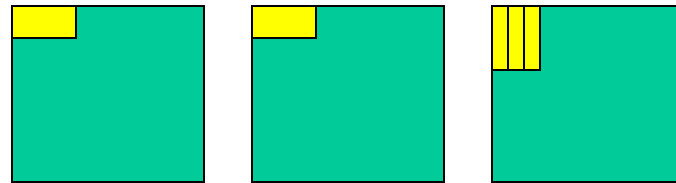


```
for (jj=0; jj<N; jj+= B)  
  for (kk=0; kk<N; kk+= B)  
    for (i=0;i<N;i++)  
      for (j=jj; j< min(jj+B,N); j++) {  
        r=0;  
        for (k=kk; k< min(kk+B,N); k++)  
          r = r + y[i][k] * z[k][j];  
        x[i][j] = x[i][j] + r;  
      }
```



Exercise

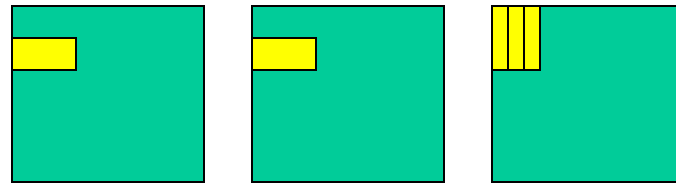
```
for (jj=0; jj<N; jj+= B)
for (kk=0; kk<N; kk+= B)
for (i=0;i<N;i++)
  for (j=jj; j< min(jj+B,N); j++) {
    r=0;
    for (k=kk; k< min(kk+B,N); k++)
      r = r + y[i][k] * z[k][j];
    x[i][j] = x[i][j] + r;
  }
```



x

y

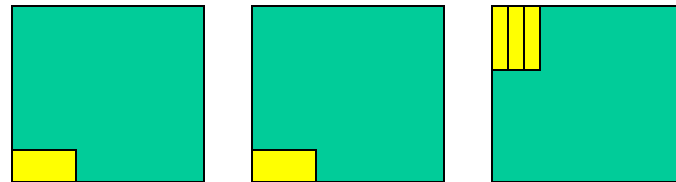
z



x

y

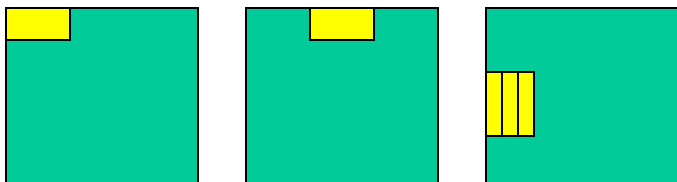
z



x

y

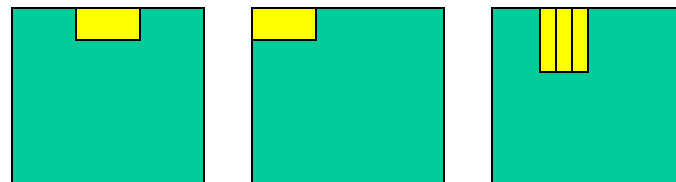
z



x

y

z



x

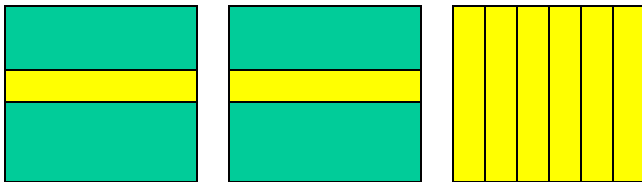
y

z

Exercise

- Original code could have $2N^3 + N^2$ memory accesses, while the new version has $2N^3/B + N^2$

```
for (i=0;i<N;i++)  
  for (j=0;j<N;j++) {  
    r=0;  
    for (k=0;k<N;k++)  
      r = r + y[i][k] * z[k][j];  
    x[i][j] = r;  
  }
```

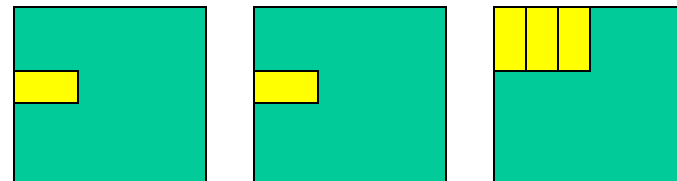


x

y

z

```
for (jj=0; jj<N; jj+= B)  
  for (kk=0; kk<N; kk+= B)  
    for (i=0;i<N;i++)  
      for (j=jj; j< min(jj+B,N); j++) {  
        r=0;  
        for (k=kk; k< min(kk+B,N); k++)  
          r = r + y[i][k] * z[k][j];  
        x[i][j] = x[i][j] + r;  
      }
```



x

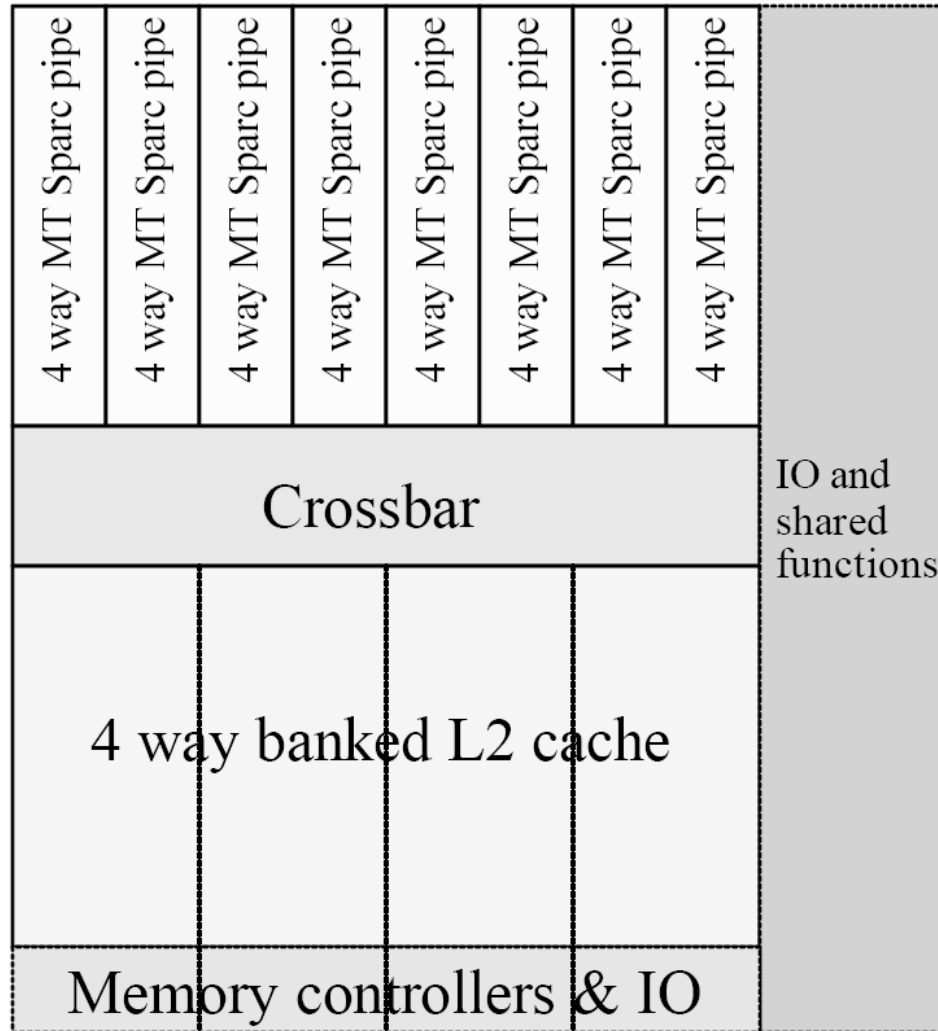
y

z

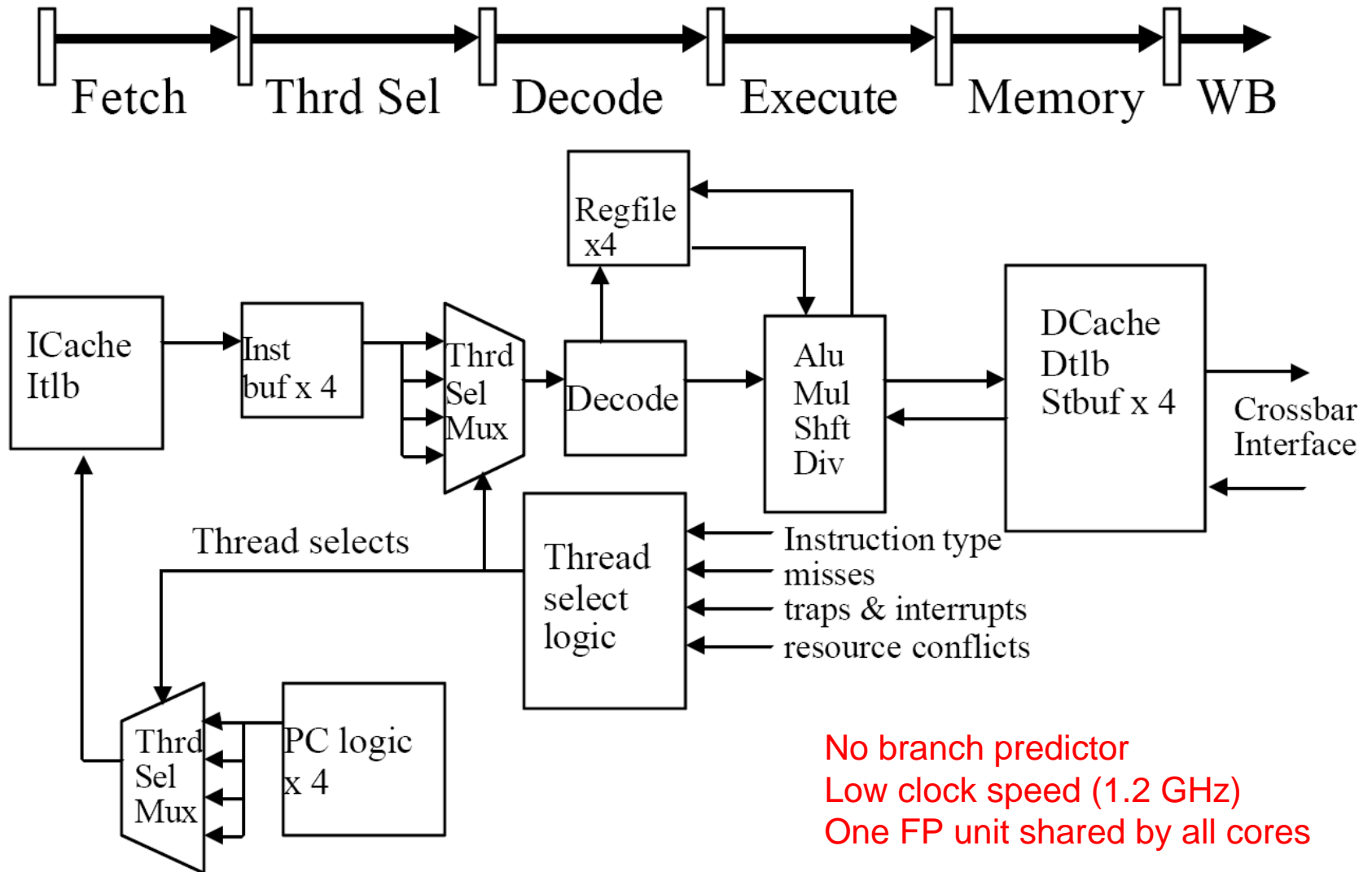
Case Study I: Sun's Niagara

- Commercial servers require high thread-level throughput and suffer from cache misses
- Sun's Niagara focuses on:
 - simple cores (low power, design complexity, can accommodate more cores)
 - fine-grain multi-threading (to tolerate long memory latencies)

Niagara Overview



SPARC Pipe

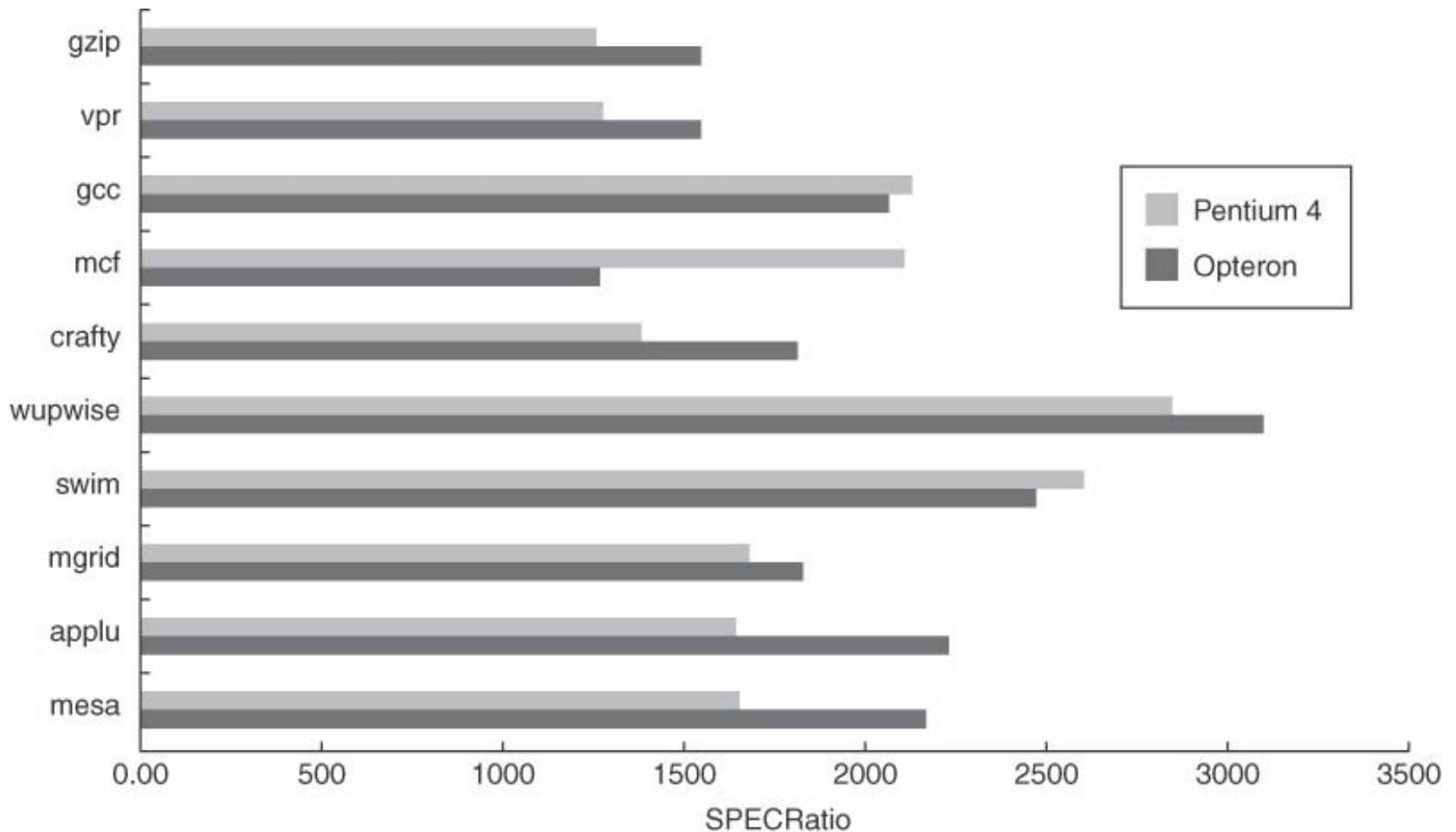


Case Study II: Intel Pentium 4

- Pursues high clock speed, ILP, and TLP
- CISC instrs are translated into micro-ops and stored in a trace cache to avoid translations every time
- Uses register renaming with 128 physical registers
- Supports up to 48 loads and 32 stores
- Rename/commit width of 3; up to 6 instructions can be dispatched to functional units every cycle
- Simple instruction has to traverse a 31-stage pipeline
- Combining branch predictor with local and global histories
- 16KB 8-way L1; 4-cyc for ints, 12-cyc for FPs; 2MB 8-way L2, 18-cyc

Clock Rate Vs. CPI: AMD Opteron Vs P4

2.8 GHz AMD Opteron vs. 3.8 GHz Intel P4: Opteron provides a speedup of 1.08



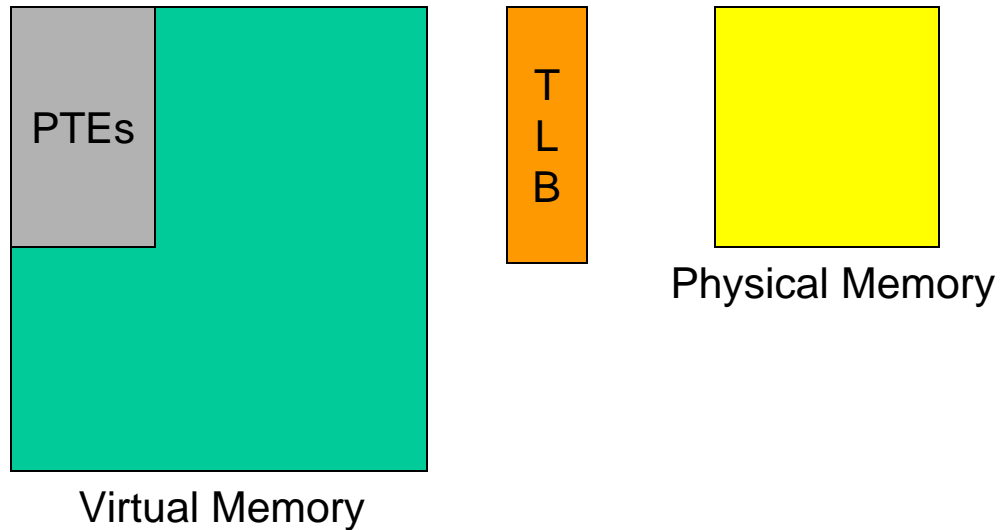
Case Study III: Intel Core Architecture

- Single-thread execution is still considered important →
 - out-of-order execution and speculation very much alive
 - initial processors will have few heavy-weight cores
- To reduce power consumption, the Core architecture (14 pipeline stages) is closer to the Pentium M (12 stages) than the P4 (30 stages)
- Many transistors invested in a large branch predictor to reduce wasted work (power)
- Similarly, SMT is also not guaranteed for all incarnations of the Core architecture (SMT makes a hotspot hotter)

Case Study IV: More Processors

- Intel Nehalem: successor to Core: to be released in late'08 with 8 cores (45 nm tech); 4-wide issue, support for SMT
- AMD Barcelona: 4 cores, issue width of 3, each core has private L1 (64 KB) and L2 (512 KB), shared L3 (2 MB), 95 W (AMD also has announcements for 3-core chips)
- Sun Niagara2: 8 threads per core, up to 8 cores, 60-123 W, 0.9-1.4 GHz, 4 MB L2 (8 banks), 8 FP units
- Sun Rock: in development (to be released in 2008), has support for “scout threads”, and transactional memory, 16 cores, 2 threads/core
- IBM Power6: 2 cores, 4.7 GHz, each core has a private 4 MB L2

Example Look-Up

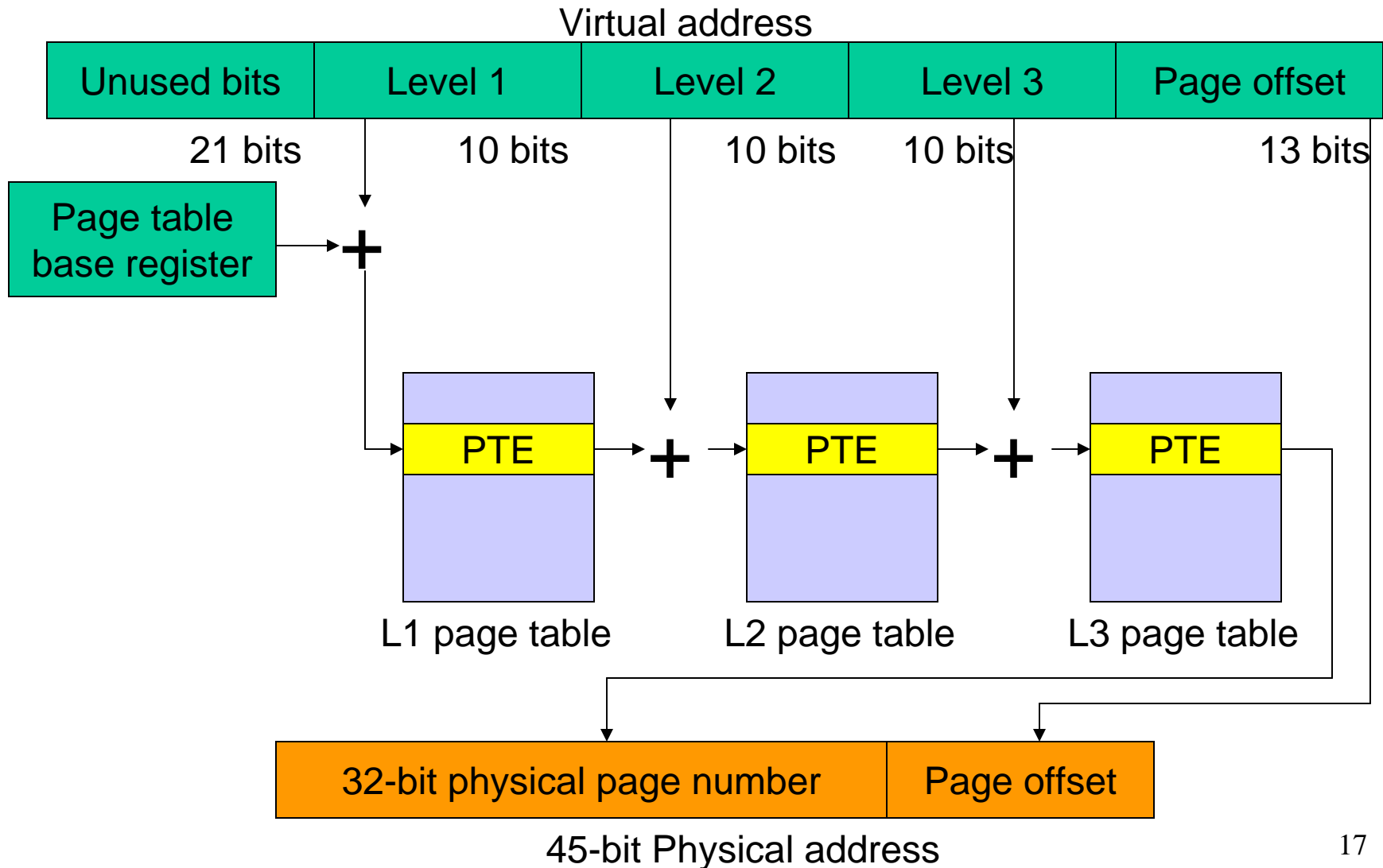


Virtual page abc \rightarrow Physical page xyz

If each PTE is 8 bytes, location of PTE
for abc is at virtual address $abc/8 = lmn$

Virtual addr lmn \rightarrow physical addr pqr

Alpha Address Mapping



Alpha Address Mapping

- Each PTE is 8 bytes – if page size is 8KB, a page can contain 1024 PTEs – 10 bits to index into each level
- If page size doubles, we need 47 bits of virtual address
- Since a PTE only stores 32 bits of physical page number, the physical memory can be addressed by at most 32 + offset
- First two levels are in physical memory; third is in virtual
- Why the three-level structure? Even a flat structure would need PTEs for the PTEs that would have to be stored in physical memory – more levels of indirection make it easier to dynamically allocate pages

Title

- Bullet