Constructive Computer Architecture

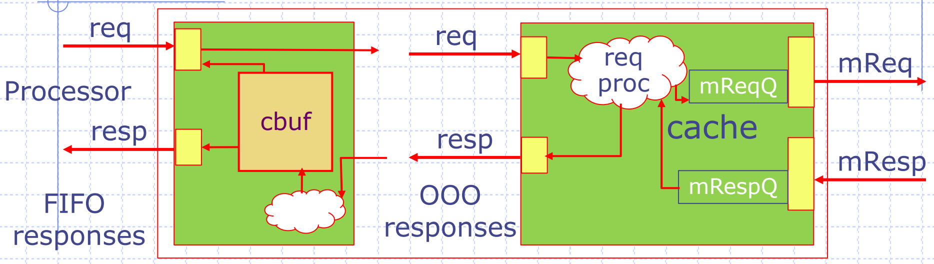
Store Buffers and Non-blocking Caches

Arvind
Computer Science & Artificial Intelligence Lab.
Massachusetts Institute of Technology

Contributors to the course material

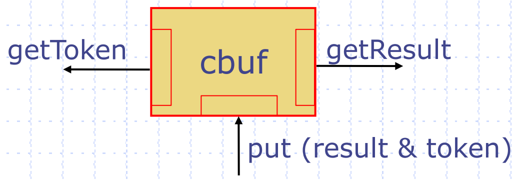
- Arvind, Rishiyur S. Nikhil, Joel Emer, Muralidaran Vijayaraghavan
- Staff and students in 6.375 (Spring 2013), 6.S195 (Fall 2012), 6.S078 (Spring 2012)
  - Asif Khan, Richard Ruhler, Sang Woo Jun, Abhinav Agarwal, Myron King, Kermin Fleming, Ming Liu, Li-Shiuan Peh
- External
  - Prof Amey Karkare & students at IIT Kanpur
  - Prof Jihong Kim & students at Seoul Nation University
  - Prof Derek Chiou, University of Texas at Austin
  - Prof Yoav Etsion & students at Technion

Non-blocking cache



- Completion buffer controls the entries of requests and ensures that departures take place in order even if loads complete out-of-order
- requests to the backend have to be tagged

Completion buffer: Interface



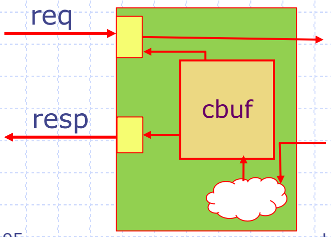
```
interface CBuffer#(type t);
  method ActionValue#(Token) getToken;
  method Action put(Token tok, t d);
  method ActionValue#(t) getResult;
endinterface
```

Concurrency requirement

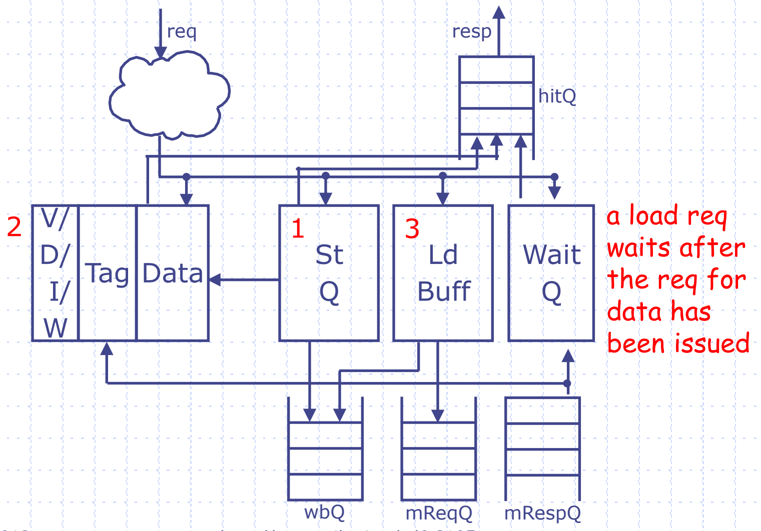
getToken < put < getResult

Non-blocking FIFO Cache

```
module mkNBFifoCache(Cache);
  CBuffer    cBuf <- mkCompletionBuffer;
  NBCache nbCache <- mkNBtaggedCache;
  rule nbCacheResponse;
  let x <- nbCache.resp;
   cBuf.put(x);
  endrule
  method Action req(MemReq x);
    let tok <- cBuf.getToken;
    nbCache.req(TaggedMemReq{req:x, tag:tok});
  endmethod
  method MemResp resp;
    let x <- cBuf.getResult
    return x
  endmethod
endmodule
```

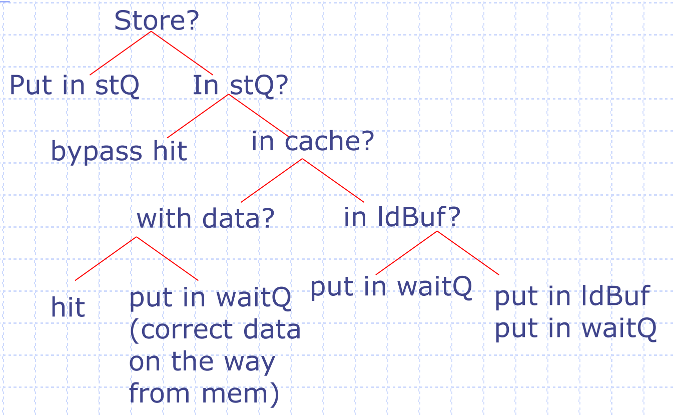


Non-blocking Cache

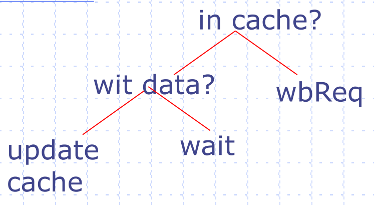


a load req waits after the req for data has been issued

Incoming req



Store buffer



Load buffer

is replacement ok?

wbReq
fill Req
replace tag
data missing

fill Req
replace tag
data missing

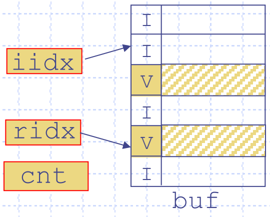
Mem Resp (line)

Update cache
Process all req for line in waitQ

Completion buffer: Implementation

A circular buffer with two pointers iidx and ridx, and a counter cnt

Elements are of Maybe type



```
module mkCompletionBuffer(CompletionBuffer#(size));
  Vector#(size, EHR#(Maybe#(t))) cb
                       <- replicateM(mkEHR(Invalid));
  Reg#(Bit#(TAdd#(TLog#(size),1)))   iidx <- mkReg(0);
  Reg#(Bit#(TAdd#(TLog#(size),1)))   ridx <- mkReg(0);
  EHR#(Bit#(TAdd#(TLog#(size),1)))    cnt <- mkEHR(0);
  Integer vsize = valueOf(size);
  Bit#(TAdd#(TLog#(size),1)) sz = fromInteger(vsize);
  rules and methods...
endmodule
```

Completion Buffer *cont*

```
method ActionValue#(t) getToken() if(cnt.r0!==sz);
  cb[iidx].w0(Invalid);
  iidx <= iidx==sz-1 ? 0 : iidx + 1;
  cnt.w0(cnt.r0 + 1);
  return iidx;
endmethod
method Action put(Token idx, t data);
  cb[idx].w1(Valid data);
endmethod
method ActionValue#(t) getResult() if(cnt.r1 !== 0
             &&&(cb[ridx].r2 matches tagged (Valid .x));
  cb[ridx].w2(Invalid);
  ridx <= ridx==sz-1 ? 0 : ridx + 1;
  cnt.w1(cnt.r1 - 1);
  return x;
endmethod
```

getToken < put < getResult