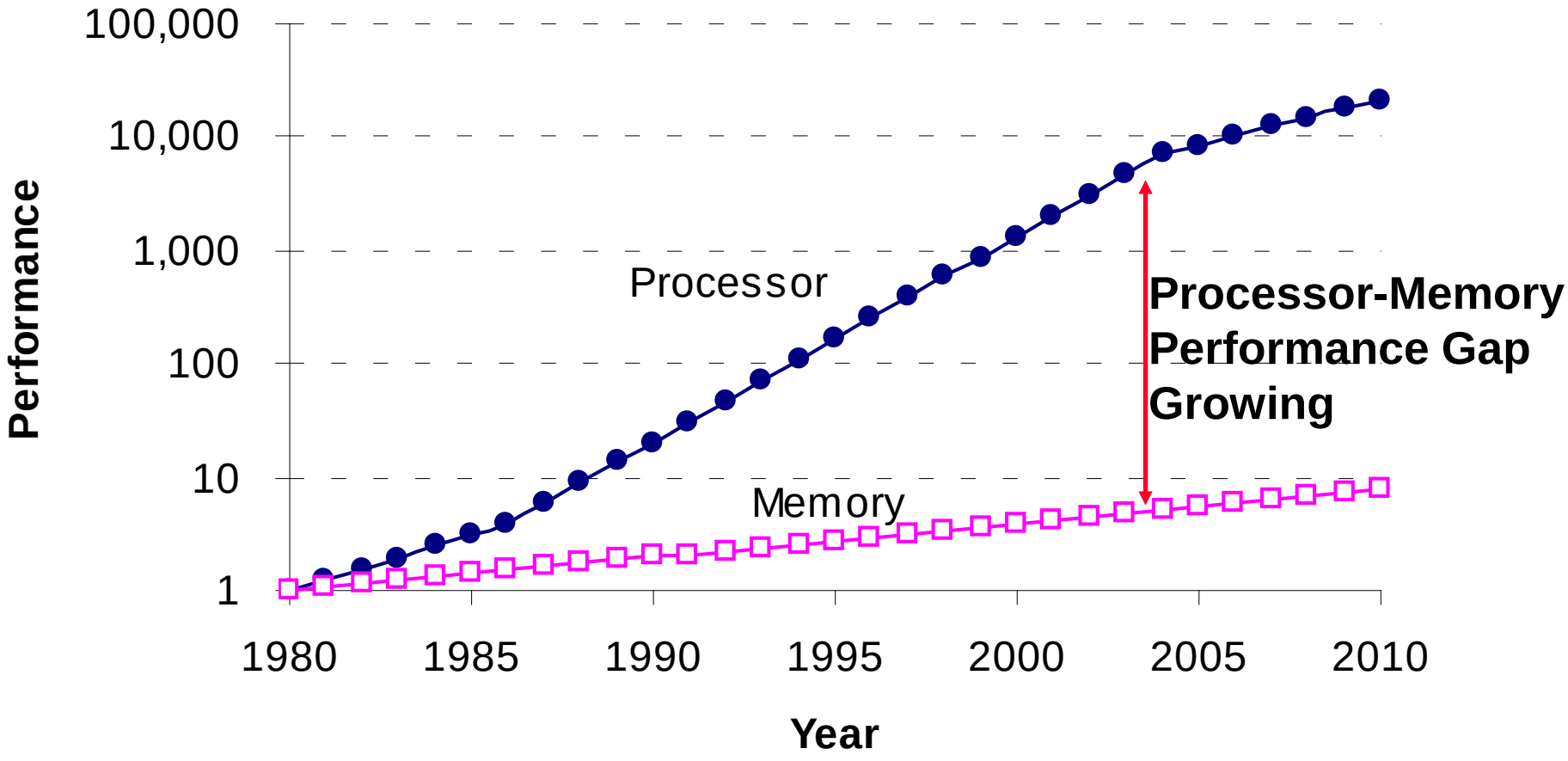
Graduate Computer Architecture

Handout 4B – Cache optimizations and inside DRAM

Outline

- **11 Advanced Cache Optimizations**
- **What inside DRAM?**
- **Summary**

Why More on Memory Hierarchy?



Review: 6 Basic Cache Optimizations

- **Reducing hit time**

1. **Giving Reads Priority over Writes**
   - **E.g., Read complete before earlier writes in write buffer**
2. **Avoiding Address Translation during Cache Indexing**

- **Reducing Miss Penalty**

3. **Multilevel Caches**

- **Reducing Miss Rate**

4. **Larger Block size (Compulsory misses)**
5. **Larger Cache size (Capacity misses)**
6. **Higher Associativity (Conflict misses)**

11 Advanced Cache Optimizations

- **Reducing hit time**

1. **Small and simple caches**
2. **Way prediction**
3. **Trace caches**

- **Increasing cache bandwidth**

4. **Pipelined caches**
5. **Multibanked caches**
6. **Nonblocking caches**

- **Reducing Miss Penalty**

7. **Critical word first**
8. **Merging write buffers**

- **Reducing Miss Rate**

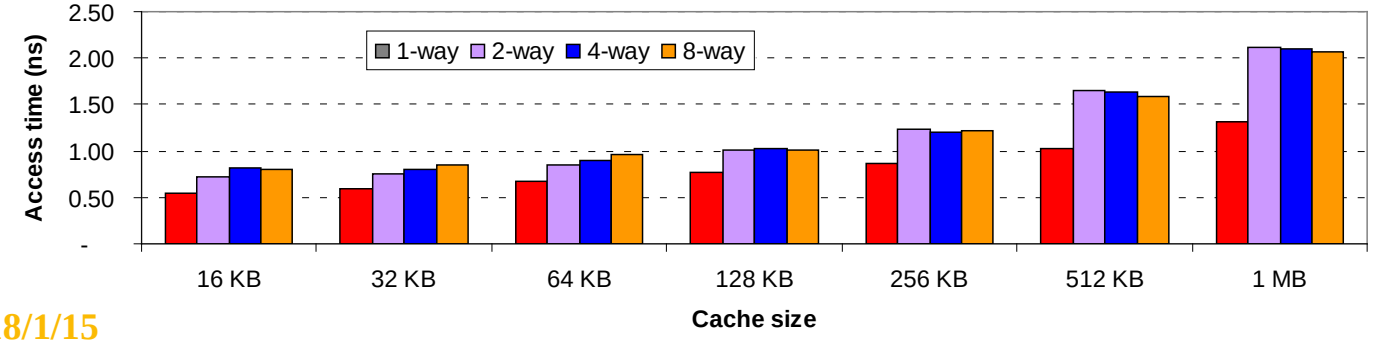
9. **Compiler optimizations**

- **Reducing miss penalty or miss rate via parallelism**

10. **Hardware prefetching**
11. **Compiler prefetching**

1. Fast Hit times via Small and Simple Caches

- Index tag memory and then compare takes time
- ⇒ Small cache can help hit time since smaller memory takes less time to index
  - E.g., L1 caches same size for 3 generations of AMD microprocessors: K6, Athlon, and Opteron
  - Also L2 cache small enough to fit on chip with the processor avoids time penalty of going off chip
- Simple ⇒ direct mapping
  - Can overlap tag check with data transmission since no choice
- Access time estimate for 90 nm using CACTI model 4.0
  - Median ratios of access time relative to the direct-mapped caches are 1.32, 1.39, and 1.43 for 2-way, 4-way, and 8-way caches



2. Fast Hit times via Way Prediction

- **How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?**
- **Way prediction: keep extra bits in cache to predict the "way," or block within the set, of next cache access.**
  - **Multiplexor is set early to select desired block, only 1 tag comparison performed that clock cycle in parallel with reading the cache data**
  - **Miss ⇒ 1st check other blocks for matches in next clock cycle**

- **Accuracy ≈ 85%**
- **Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles**
  - **Used for instruction caches vs. data caches**

3. Fast Hit times via Trace Cache (Pentium 4 only; and last time?)

- **Find more instruction level parallelism? How avoid translation from x86 to microops?**
- **Trace cache in Pentium 4**

1. **Dynamic traces of the executed instructions vs. static sequences of instructions as determined by layout in memory**
   - **Built-in branch predictor**
2. **Cache the micro-ops vs. x86 instructions**
   - **Decode/translate from x86 to micro-ops on trace cache miss**

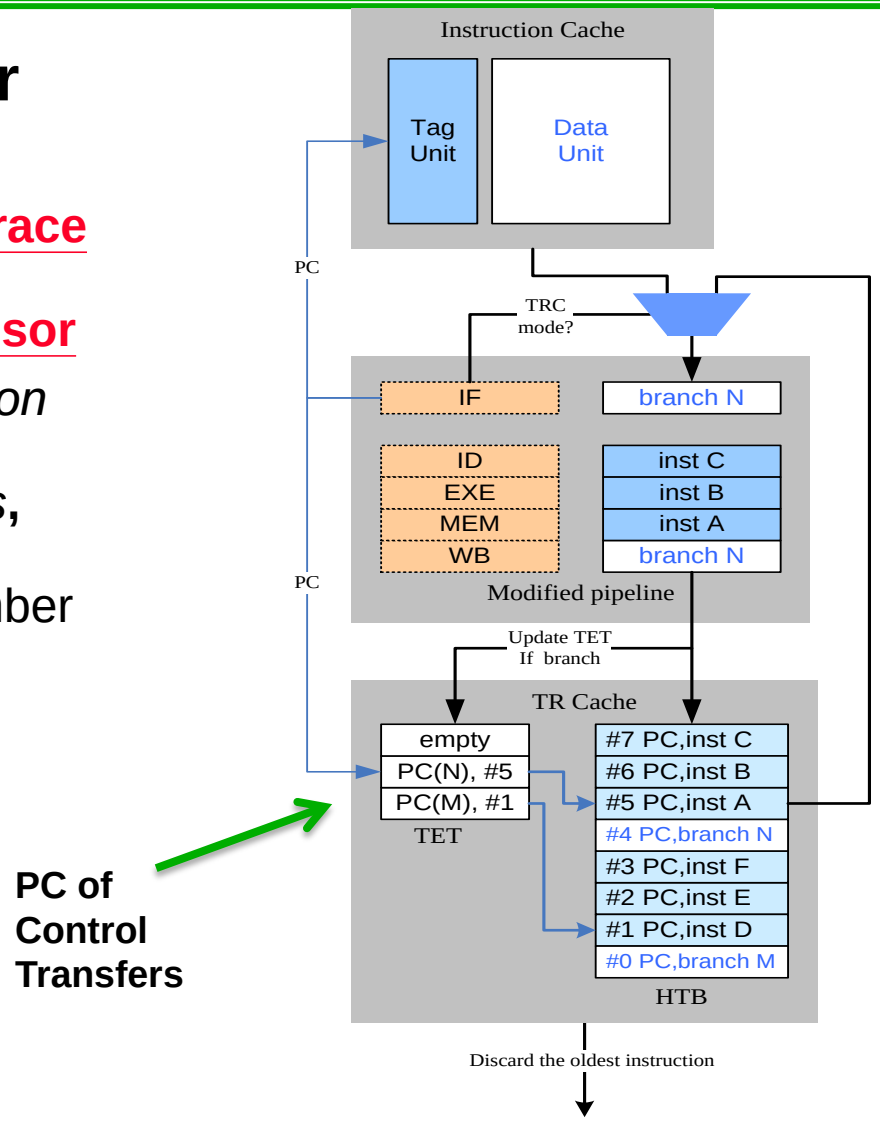
\+ **1. ⇒ better utilize long blocks (don't exit in middle of block, don't enter at label in middle of block)**

\- **1. ⇒ complicated address mapping since addresses no longer aligned to power-of-2 multiples of word size**

\- **1. ⇒ instructions may appear multiple times in multiple dynamic traces due to different branch outcomes**

Our work: index-based FIFO trace

- **Reuse trace after execution:**
  - **Energy-efficient Trace Reuse Cache for Embedded Processor**
  - *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 19, No. 9, pp. 1681-1694, September 2011.



4: Increasing Cache Bandwidth by Pipelining

- **Pipeline cache access to maintain bandwidth, but higher latency**
- **Instruction cache access pipeline stages:**

  1: Pentium

  2: Pentium Pro through Pentium III

  4: Pentium 4

- **⇒ greater penalty on mispredicted branches**
- **⇒ more clock cycles between the issue of the load and the use of the data**

Pipeline Cache Access

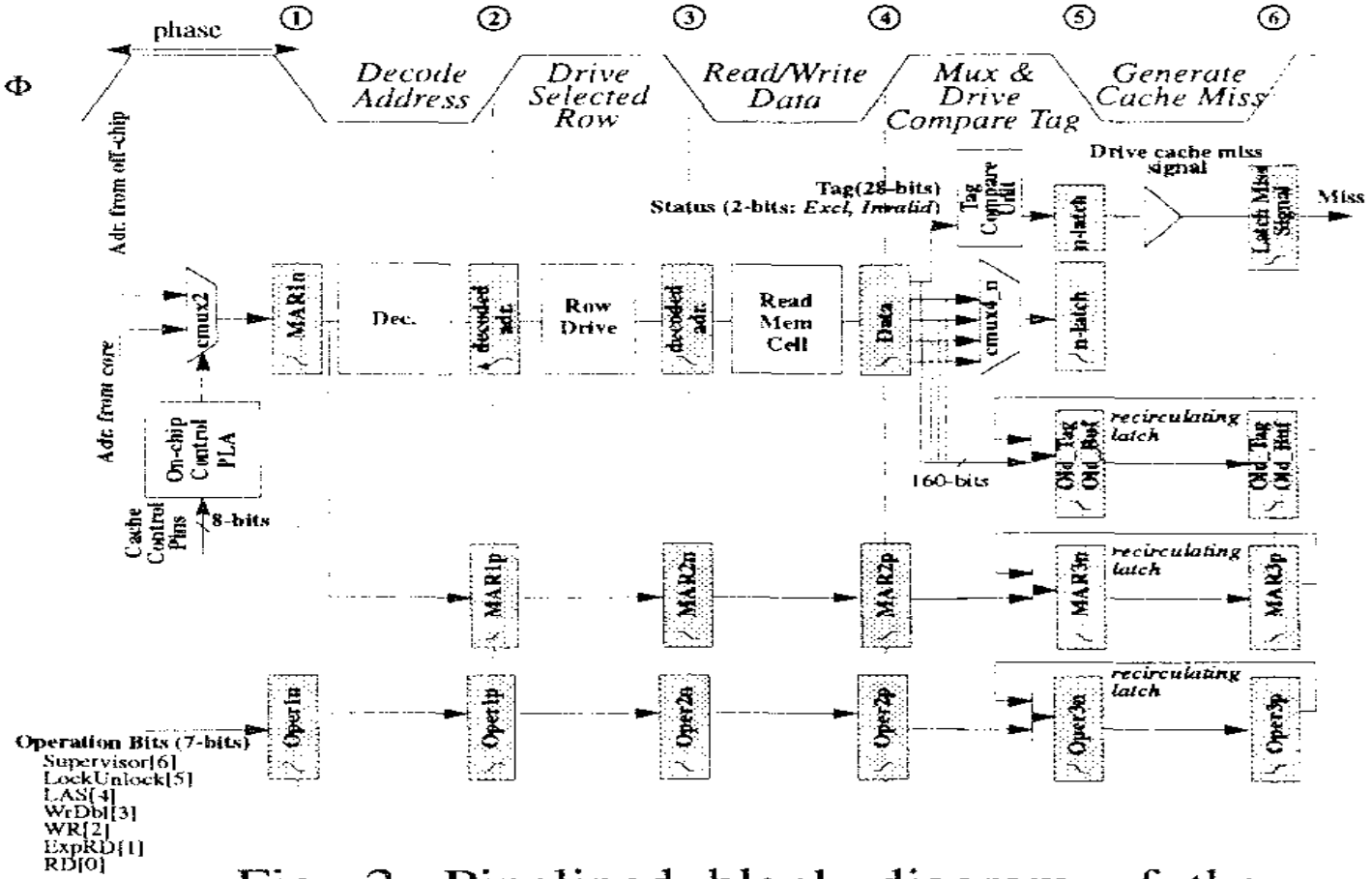


Fig. 2. Pipelined block diagram of the major functional units of the Ucache.

5. Increasing Cache Bandwidth: Non-Blocking Caches

- ***Non-blocking cache*** **or** ***lockup-free cache*** **allow data cache to continue to supply cache hits during a miss**
  - **requires Memory Status Holding Registers or out-of-order execution**
  - **requires multi-bank memories**
- **"*hit under miss*" reduces the effective miss penalty by working during miss vs. ignoring CPU requests**
- **"*hit under multiple miss*" or "*miss under miss*" may further lower the effective miss penalty by overlapping multiple misses**
  - **Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses**
  - **Requires muliple memory banks (otherwise cannot support)**
  - **Penium Pro allows 4 outstanding memory misses**

6: Increasing Cache Bandwidth via Multiple Banks

- **Rather than treat the cache as a single monolithic block, divide into independent banks that can support simultaneous accesses**
  - **E.g.,T1 ("Niagara") L2 has 4 banks**
- **Banking works best when accesses naturally spread themselves across banks ⇒ mapping of addresses to banks affects behavior of memory system**
- **Simple mapping that works well is "sequential interleaving"**
  - **Spread block addresses sequentially across banks**
  - **E,g, if there 4 banks, Bank 0 has all blocks whose address modulo 4 is 0; bank 1 has all blocks whose address modulo 4 is 1; …**

7. Reduce Miss Penalty: Early Restart and Critical Word First

- **Don’t wait for full block before restarting CPU**
- ***Early restart*—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution**
  - Spatial locality ⇒ tend to want next sequential word
- ***Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block**
  - **Long blocks more popular today ⇒ Critical Word 1st Widely used**

block

8. Merging Write Buffer to Reduce Miss Penalty

- Write buffer to allow processor to continue while waiting to write to memory
- If buffer contains modified blocks, the addresses can be checked to see if address of new data matches the address of a valid write buffer entry
- If so, new data are combined with that entry
- Increases block size of write for write-through cache of writes to sequential words, bytes since multiword writes more efficient to memory
- The Sun T1 (Niagara) processor, among many others, uses write merging

9. Reducing Misses by Compiler Optimizations

- **Instructions**
  - **Reorder procedures in memory so as to reduce conflict misses**
  - **Profiling to look at conflicts(using tools they developed)**
- **Data**
  - ***Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays**
  - ***Loop Interchange*: change nesting of loops to access data in order stored in memory**
  - ***Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap**
  - ***Blocking*: Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows**

Merging Arrays Example

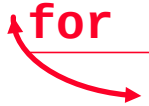
```
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array of stuctures */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```

Reducing conflicts between val & key;
improve spatial locality

Loop Interchange Example

```
/* Before */
for (k = 0; k < 100; k = k+1)
  for (j = 0; j < 100; j = j+1)
       for (i = 0; i < 5000; i = i+1)
              x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
  for (i = 0; i < 5000; i = i+1)
       for (j = 0; j < 100; j = j+1)
              x[i][j] = 2 * x[i][j];
```



Sequential accesses instead of striding through memory every 100 words; improved spatial locality

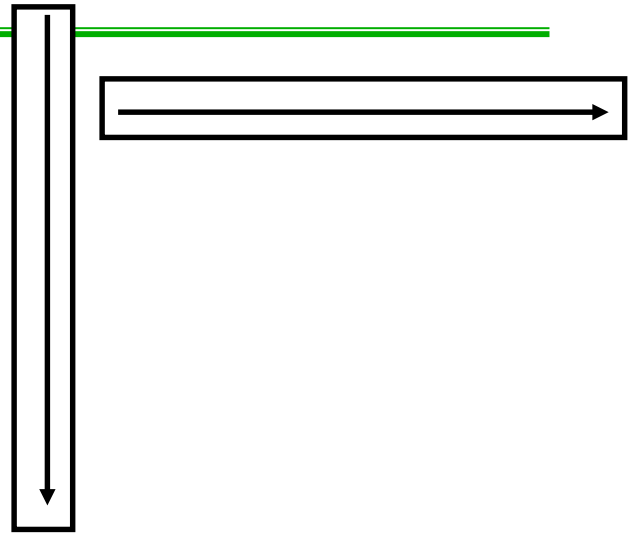
Loop Fusion Example

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
       a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
       d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {    a[i][j] = 1/b[i][j] * c[i][j];
       d[i][j] = a[i][j] + c[i][j];}
```

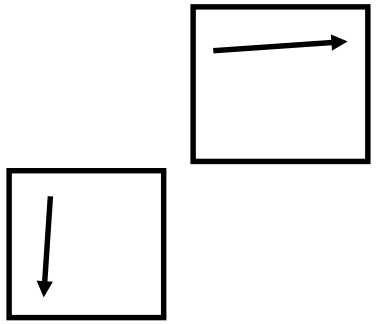
2 misses per access to a & c vs. one miss per access; improve spatial locality

Blocking Example

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
     {r = 0;
      for (k = 0; k < N; k = k+1){
        r = r + y[i][k]*z[k][j];};
      x[i][j] = r;
     };
```



- Two Inner Loops:
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]
- Idea: compute on BxB submatrix that fits

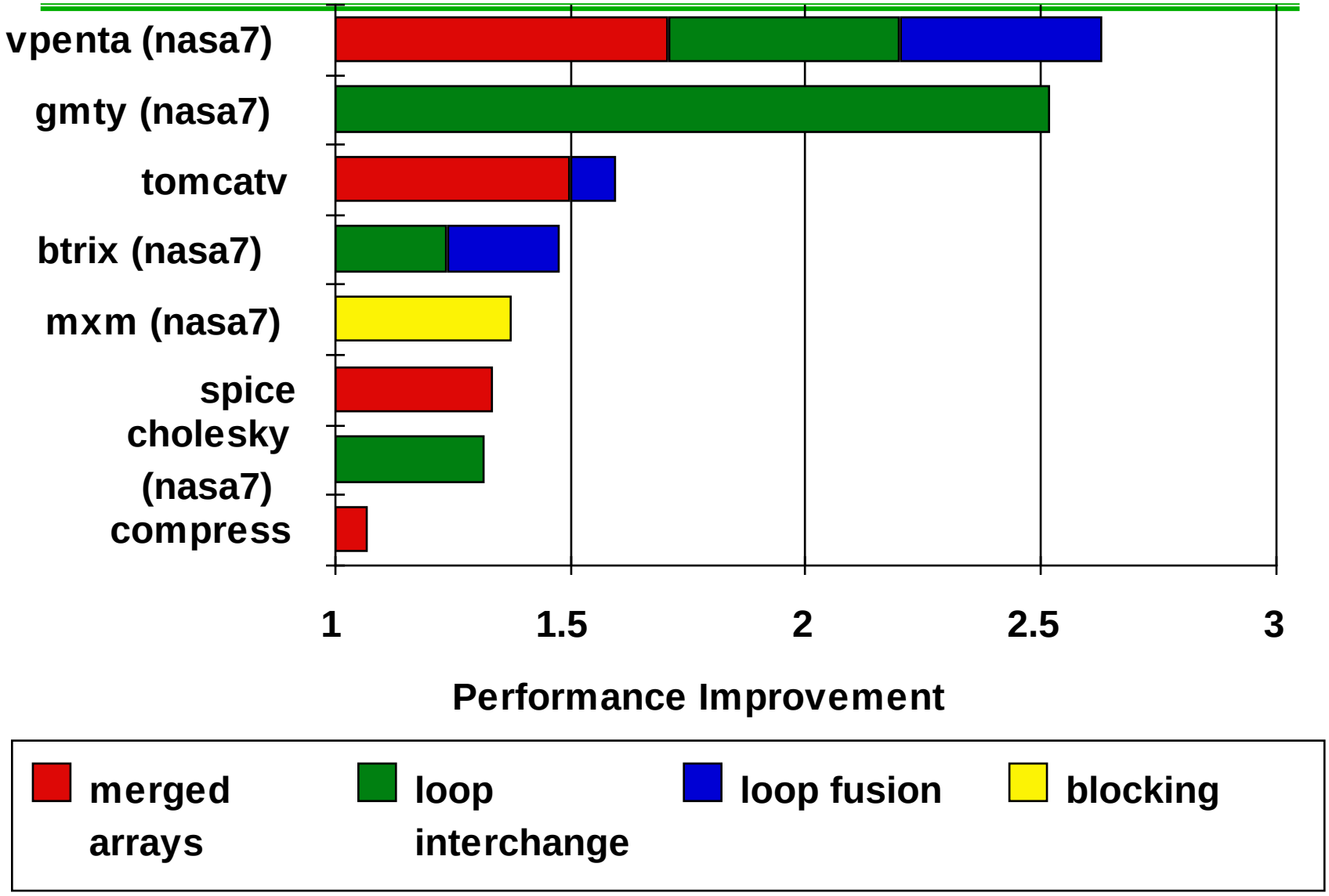


Blocking Example

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
   for (j = jj; j < min(jj+B-1,N); j = j+1)
     {r = 0;
      for (k = kk; k < min(kk+B-1,N); k = k+1) {
        r = r + y[i][k]*z[k][j];};
      x[i][j] = x[i][j] + r;
     };
```

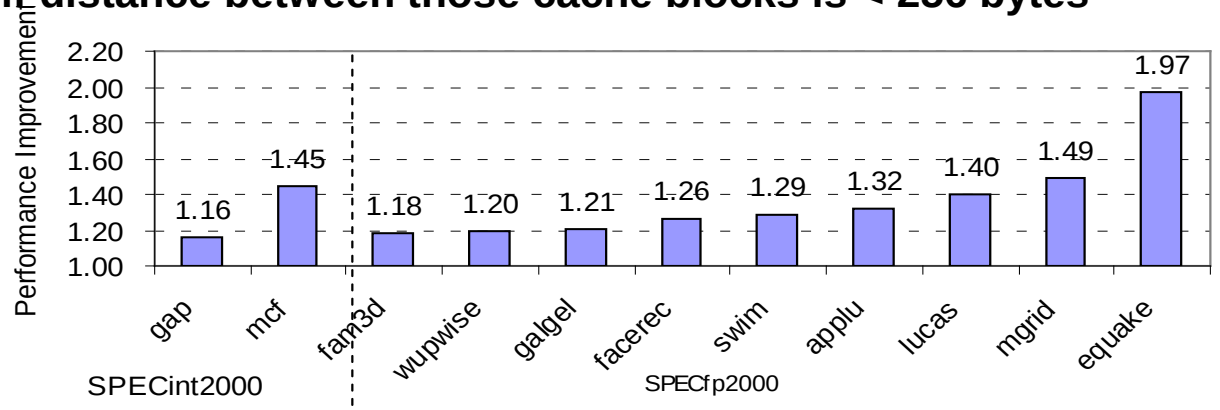
- B called *Blocking Factor*

Summary of Compiler Optimizations to Reduce Cache Misses (by hand)



10. Reducing Misses by Hardware Prefetching of Instructions & Data

- **Prefetching relies on having extra memory bandwidth that can be used without penalty**
- **Instruction Prefetching**
  - **Typically, CPU fetches 2 blocks on a miss: the requested block and the next consecutive block.**
  - **Requested block is placed in instruction cache when it returns, and prefetched block is placed into instruction stream buffer**
- **Data Prefetching**
  - **Pentium 4 can prefetch data into L2 cache from up to 8 streams from 8 different 4 KB pages**
  - **Prefetching invoked if 2 successive L2 cache misses to a page, if distance between those cache blocks is < 256 bytes**



11. Reducing Misses by Software Prefetching Data

- **Data Prefetch**
  - **Load data into register (HP PA-RISC loads)**
  - **Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)**
  - **Special prefetching instructions cannot cause faults; a form of speculative execution**
- **Issuing Prefetch Instructions takes time**
  - **Is cost of prefetch issues < savings in reduced misses?**
  - **Higher superscalar reduces difficulty of issue bandwidth**

Technique	Hit Time	Band-width	Miss penalty	Miss rate	HW cost/ complexity	Comment
Small and simple caches	+			–	0	Trivial; widely used
Way-predicting caches	+				1	Used in Pentium 4
Trace caches	+				3	Used in Pentium 4
Pipelined cache access	–	+			1	Widely used
Nonblocking caches		+	+		3	Widely used
Banked caches		+			1	Used in L2 of Opteron and Niagara
Critical word first and early restart			+		2	Widely used
Merging write buffer			+		1	Widely used with write through
Compiler techniques to reduce cache misses				+	0	Software is a challenge; some computers have compiler option
Hardware prefetching of instructions and data			+	+	2 instr., 3 data	Many prefetch instructions; AMD Opteron prefetches data
Compiler-controlled prefetching			+	+	3	Needs nonblocking cache; in many CPUs

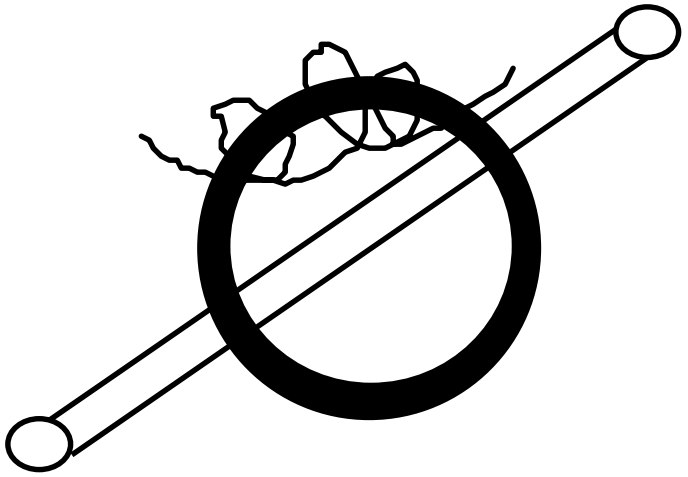
Main Memory Background

- **Performance of Main Memory:**
  - **Latency: Cache Miss Penalty**
    - » ***Access Time*: time between request and word arrives**
    - » ***Cycle Time*: time between requests**
  - **Bandwidth: I/O & Large Block Miss Penalty (L2)**
- **Main Memory is *DRAM*: Dynamic Random Access Memory**
  - **Dynamic since needs to be refreshed periodically (8 ms, 1% time)**
  - **Addresses divided into 2 halves (Memory as a 2D matrix):**
    - » ***RAS* or *Row Access Strobe***
    - » ***CAS* or *Column Access Strobe***
- **Cache uses *SRAM*: Static Random Access Memory**
  - **No refresh (6 transistors/bit vs. 1 transistor**

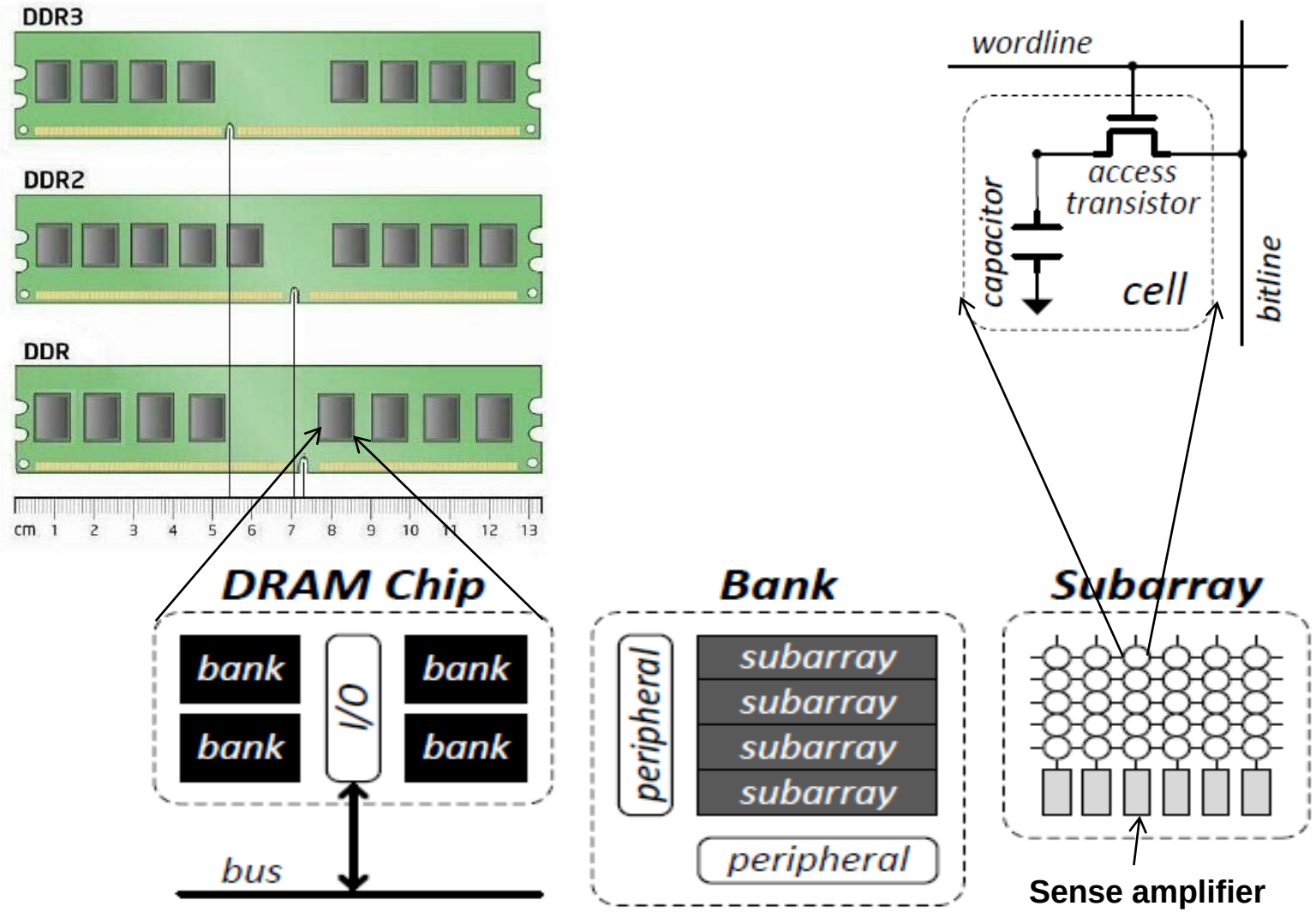
Main Memory Deep Background

"Out-of-Core", "In-Core," "Core Dump"?

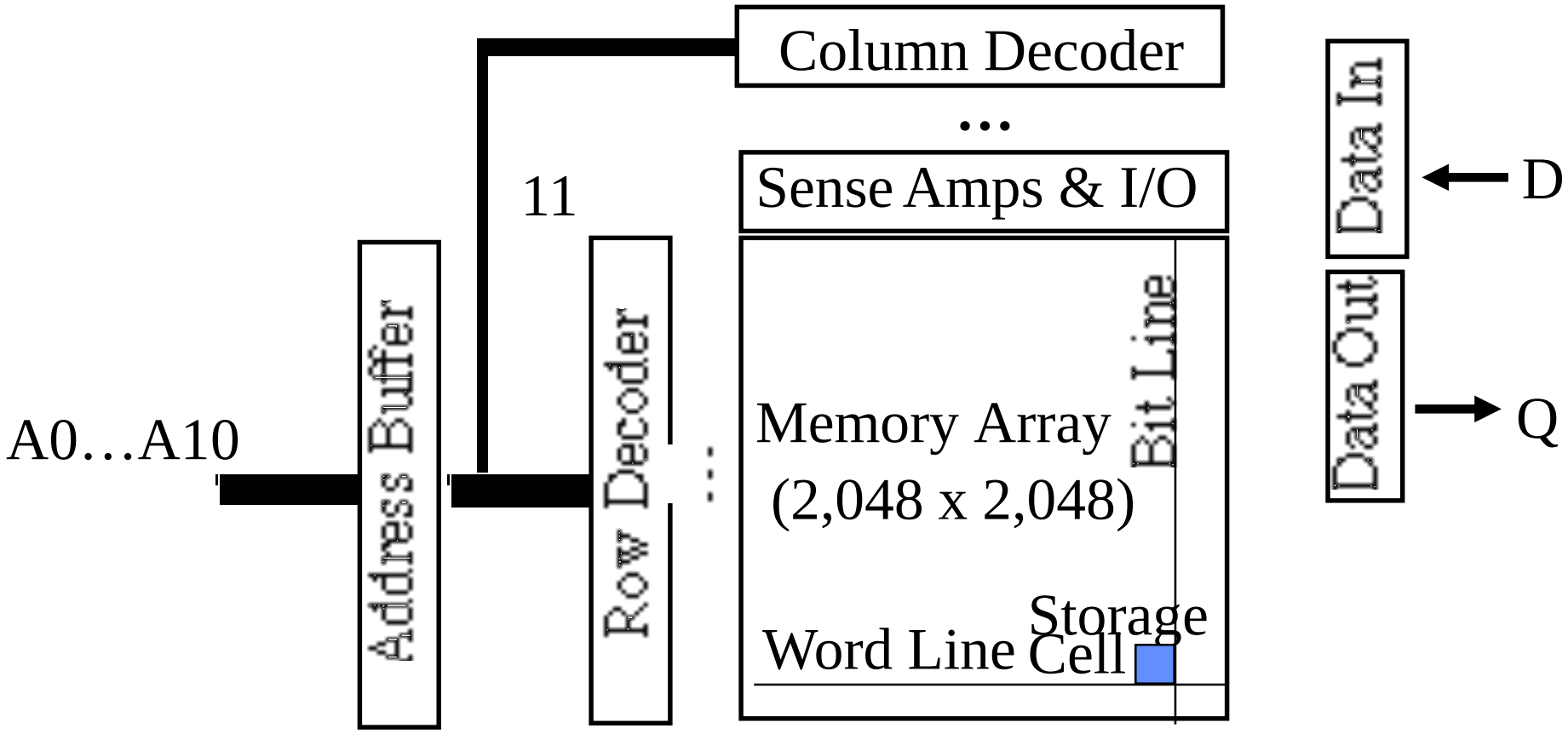
- **"Core memory"?**
- **Non-volatile, magnetic**
- **Lost to 4 Kbit DRAM (today using 512Mbit DRAM)**
- **Access time 750 ns, cycle time 1500-3000 ns**



Structure: DRAM from bit to chip



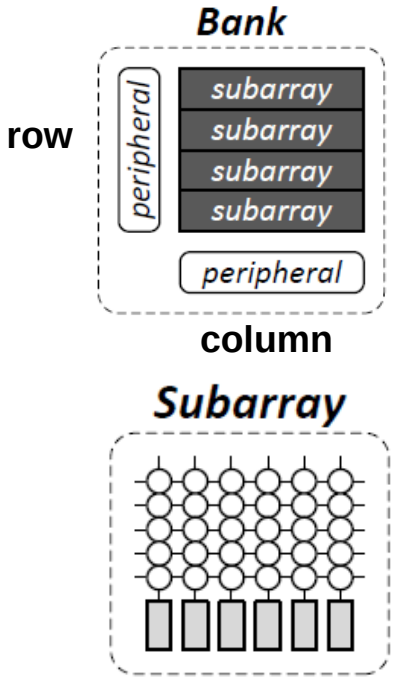
DRAM logical organization (4 Mbit)



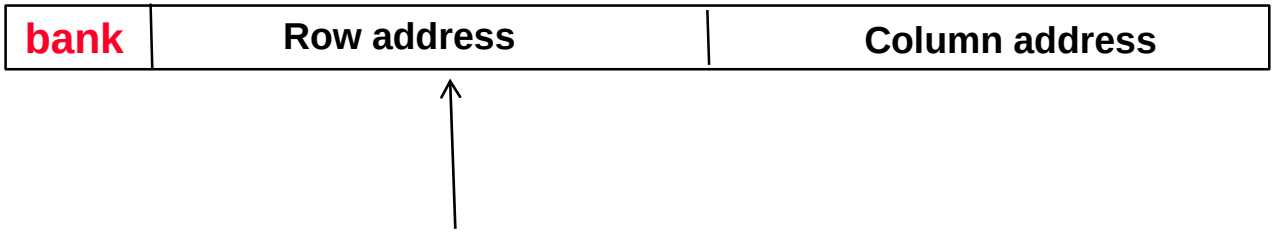
- **Square root of bits per RAS/CAS**

Addressing a DRAM chip

- A DRAM chip composes of several banks, e.g., 8 banks.
- A bank is a set of cells that share peripheral circuitry such as address decoder (row and column)
- A subarray, which is a two-dimensional array, is a set of cells that share bitlines and sense amplifiers

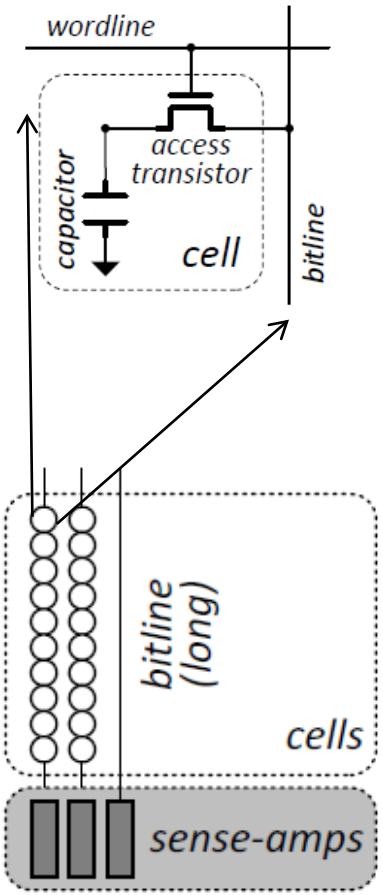


Where are the rest of address bits ?



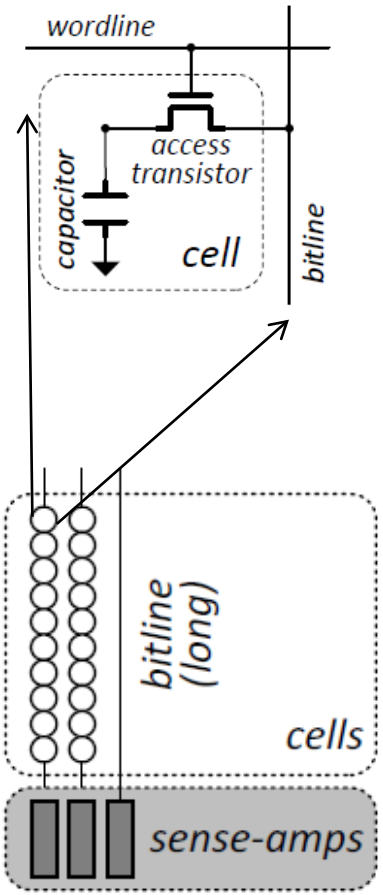
Basic: bitline and wordline

- Cell, bitline, wordline, sense amp.
  - A cell consists of a capacitor which stores electrical charge and a transistor which is turned on or turned off to connect or disconnect the capacitor to the wire called bitline.
  - Electrical change on a capacitor-based cell represents the bit value.
  - Many cells connected to one bitline to share one sense AMP.
  - One of the cells on a bitline is turned on by asserting its wordline (row address, in general).



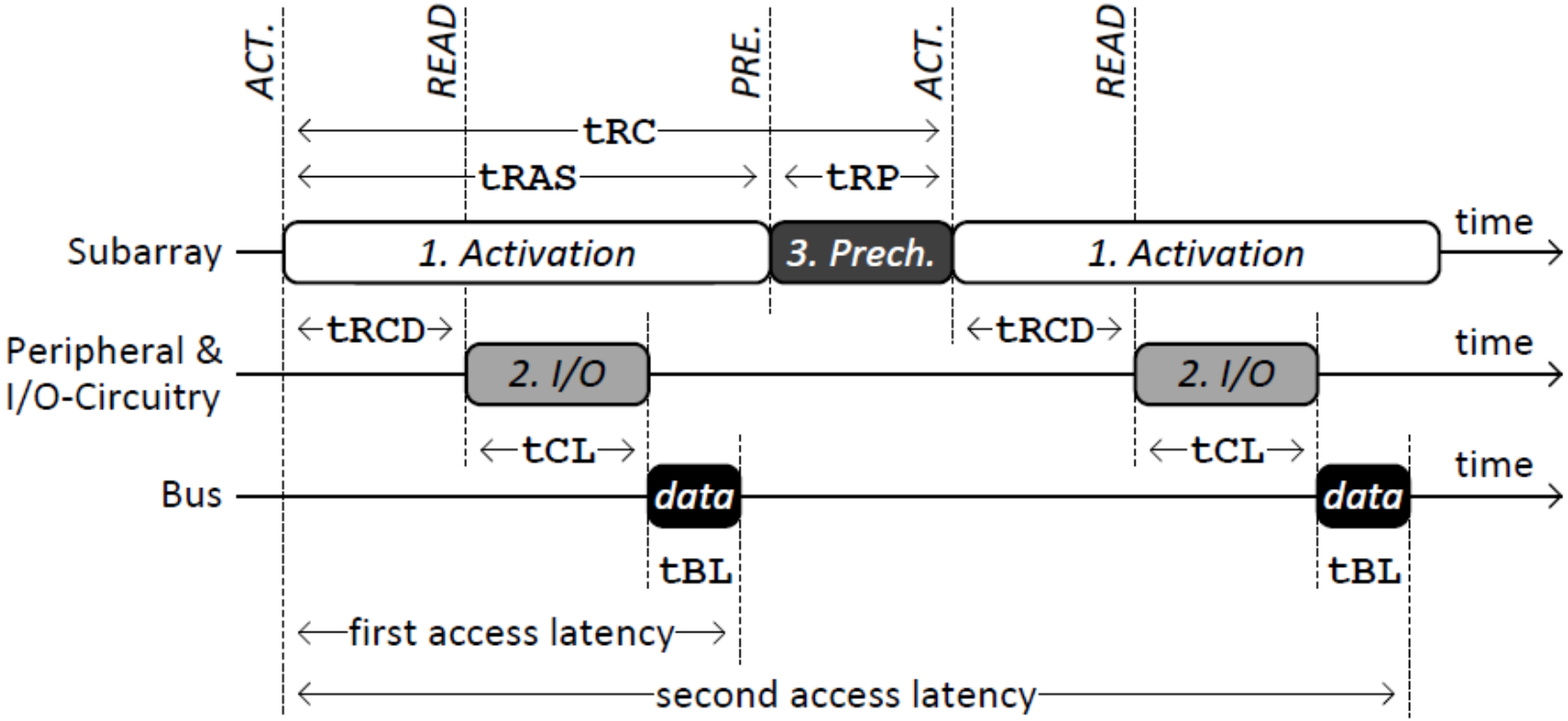
Basic: current (charge) sensing

- **Sense amp.**
  - **Due to the small size of the capacitor, a sense-amplifier is required to sense the small amount of charge held by the cell and amplify it to a full logic value.**
  - **A sense amplifier is up to three orders of magnitude larger than a cell. (10^3)**
  - **A bitline may connect up to 512 cells, i.e. 512 rows**



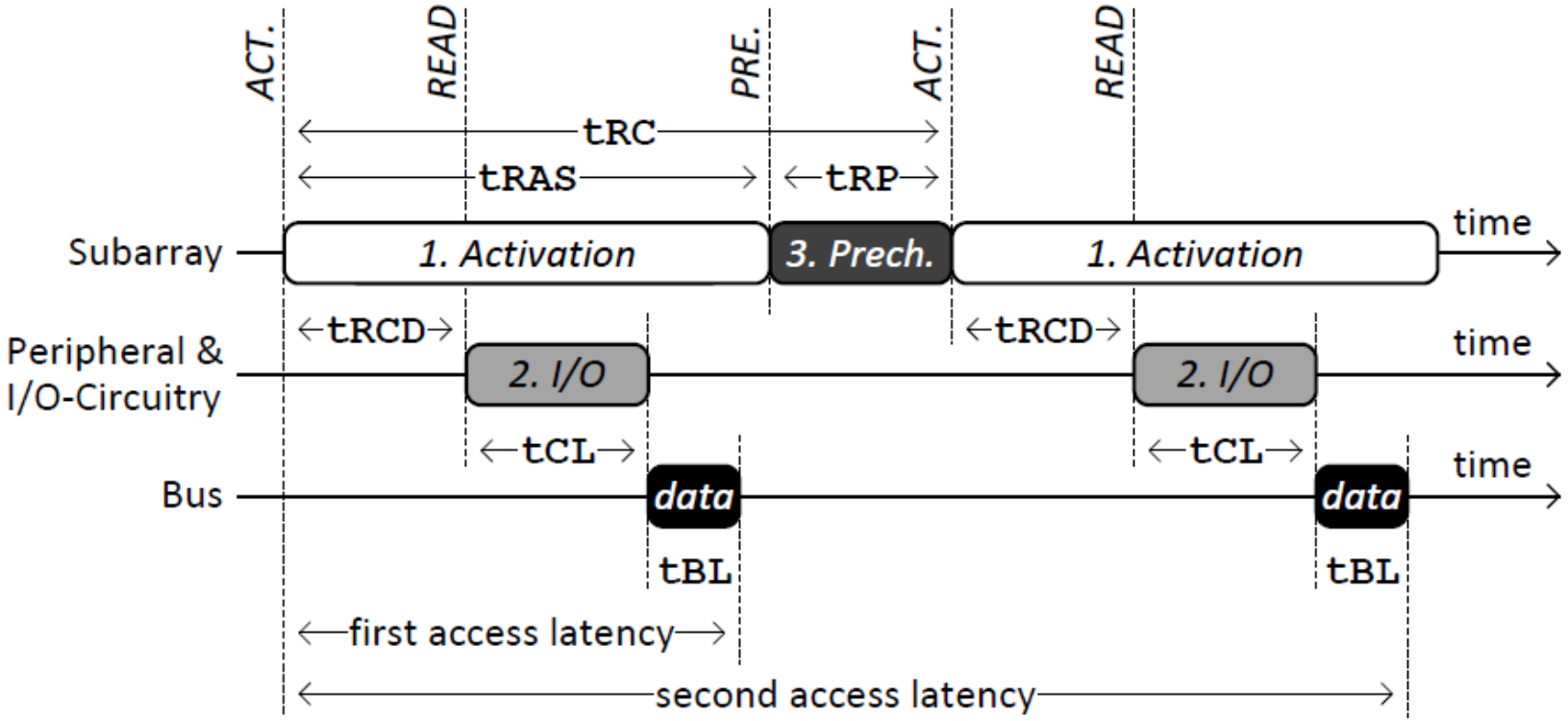
Timeline of accessing DRAM (1)

- **3 phases: activation, I/O, and precharging**
  - **1. Activation: a wordline within a subarray is asserted, connecting a row of cells to the bitlines, then data (the entire row) are copied onto the sense amps of the subarray.**



Timeline of accessing DRAM (2)

- **3 phases: activation, I/O, and precharging**
  - **2. I/O phase: the data in the sense amps are transferred through the peripheral to the DRAM's I/O circuitry and onto the memory bus**



Timeline of accessing DRAM (3)

- **3 phases: activation, I/O, and precharge**
  - **3. precharge: the raised wordline in the subarray is lowered, disconnecting the cells from the bitlines, also the sense amps and bitlines are initialized.**
  - **Precharge is used to get rid (charge) the parasitic capacitances so you would not lose any charge (ensuring the full voltage swing) during the read/write cycles.**
  - **Parasitic capacitance is the extra capacitance which every conductors possess, which means all of the wires and conductive pathways have those capacitance. Those extra capacitance, in turn, leech up some of the charge that is being sent to a capacitor, which in turn creates a delay to the capacitor getting a full charge.**
  - **Without the precharge stage, and ignoring the delay it causes, the cell that is supposed to be 'written' will have less charge than it is supposed to be, since the parasitic capacitance took them.**
  - **https://www.quora.com/SDRAM-What-does-it-mean-to-precharge-the-banks-to-get-into-the-idle-state**

Fallacy: DDR is high-speed DRAM

- **DDR3-1066**
  - Memory clock: 133 Mhz
  - I/O bus clock: 533 Mhz
  - Data rate 1066 MT
- Trick for DDR
  - Memory latency is still long
  - But transfer rate is increased.

Phase	Commands	Name	Value
1	ACT → READ ACT → WRITE	tRCD	15ns
	ACT → PRE	tRAS	37.5ns
2	READ → *data* WRITE → *data*	tCL tCWL	15ns 11.25ns
	data	tBL	7.5ns
3	PRE → ACT	tRP	15ns
1 & 3	ACT → ACT	tRC (tRAS+tRP)	52.5ns

Quest for DRAM Performance

1. **Fast Page mode**
   - Add timing signals that allow repeated accesses to row buffer without another row access time
   - Such a buffer comes naturally, as each array will buffer 1024 to 2048 bits for each access
2. **Synchronous DRAM (SDRAM)**
   - Add a clock signal to DRAM interface, so that the repeated transfers would not bear overhead to synchronize with DRAM controller
3. **Double Data Rate (DDR SDRAM)**
   - Transfer data on both the rising edge and falling edge of the DRAM clock signal ⇒ doubling the peak data rate
   - DDR2 lowers power by dropping the voltage from 2.5 to 1.8 volts + offers higher clock rates: up to 400 MHz
   - DDR3 drops to 1.5 volts + higher clock rates: up to 800 MHz

- **Improved Bandwidth, not Latency**

DRAM Optimizations

- **Shorter bitlines: reduce access latency but need more sense amplifiers, as a result, higher cost per bit**
- **Cached DRAM: add SRAM cache to DRAM chips. Need more area and transfer channel btw SRAM and DRAM**
- **Increased DRAM parallelism: parallelize accesses to different sub-array within a bank. Reduce the frequency of row buffer conflicts**
- **DRAM controller optimizations: improve row buffer locality by rescheduling accesses**

DRAM name based on Peak Chip Transfers / Sec
DIMM name based on Peak DIMM MBytes / Sec

Stan-dard	Clock Rate (MHz)	M transfers / second	DRAM Name	Mbytes/s/ DIMM	DIMM Name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10664	PC10700
DDR3	800	1600	DDR3-1600	12800	PC12800

x 2

x 8

Need for Error Correction!

- **Motivation:**
  - **Failures/time *proportional* to number of bits!**
  - **As DRAM cells shrink, more vulnerable**
- **Went through period in which failure rate was low enough without error correction that people didn't do correction**
  - **DRAM banks too large now**
  - **Servers always corrected memory systems**
- **Basic idea: add redundancy through parity bits**
  - **Common configuration: Random error correction**
    - » **SEC-DED (single error correct, double error detect)**
    - » **One example: 64 data bits + 8 parity bits (11% overhead)**
  - **Really want to handle failures of physical components as well**
    - » **Organization is multiple DRAMs/DIMM, multiple DIMMs**
    - » **Want to recover from failed DRAM and failed DIMM!**
    - » **"Chip kill" handle failures width of single DRAM chip**

And in Conclusion …

- **Memory wall inspires optimizations since so much performance lost there**
  - **Reducing hit time: Small and simple caches, Way prediction, Trace caches**
  - **Increasing cache bandwidth: Pipelined caches, Multibanked caches, Nonblocking caches**
  - **Reducing Miss Penalty: Critical word first, Merging write buffers**
  - **Reducing Miss Rate: Compiler optimizations**
  - **Reducing miss penalty or miss rate via parallelism: Hardware prefetching, Compiler prefetching**
- **DRAM – Continuing Bandwidth innovations: Fast page mode, Synchronous, Double Data Rate**