Dynamic Memory Systems Design

- **Introduction and Overview**
- **Dram Technology**
- **Conventional Asynchronous Dram Timing Analysis:**
  - **Ras*/Cas* Signalling,**
  - **Access and Cycle Times.**

- **Asynchronous Dram Controller for a 68000 Based System.**
- **Refreshing Concepts.**
- **Fast Page Mode Drams,**
- **Synchronous Dram (SDRAM)**
- **Double Data Rate Synchronous Drams (DDR-SDRAM)**

Introduction and Overview

Introduction

- Dynamic ram (DRAM) is an alternative to SRAM for random access, volatile R/W memory.
- Mainly used in high performance computers, such as PC's or workstations.

Benefits of Dram over Sram are

- **Cost, Storage Capacity, Power and Packaging Densities** :

  - Beyond a certain size it's much cheaper to build large memory systems around Dram than Sram.
  - This is because a dram cell is typically 18 x smaller than the equivalent sram cell, requiring 1 transistor per bit rather than 6 (+interconnect)
  - This results in less power consumption and more silicon available for memory cells leading to higher storage densities than to Sram.
  - Packaged dram chips are also smaller than their equivalent Sram packages as they utilize a multiplexed address bus which also leads to a reduction in PCB size.

Introduction and Overview

Drawbacks of using Drams:

- **Complex Interfacing**: Dram require a dedicated controller that can take a CPU address and multiplex it (split into two halves) onto the Dram address lines. The timing of signals is also more complex.
- **Need for Refreshing**: Due to their inherent design characteristics, Drams have a nasty habit of *forgetting* their data if they are not periodically refreshed by external circuitry.

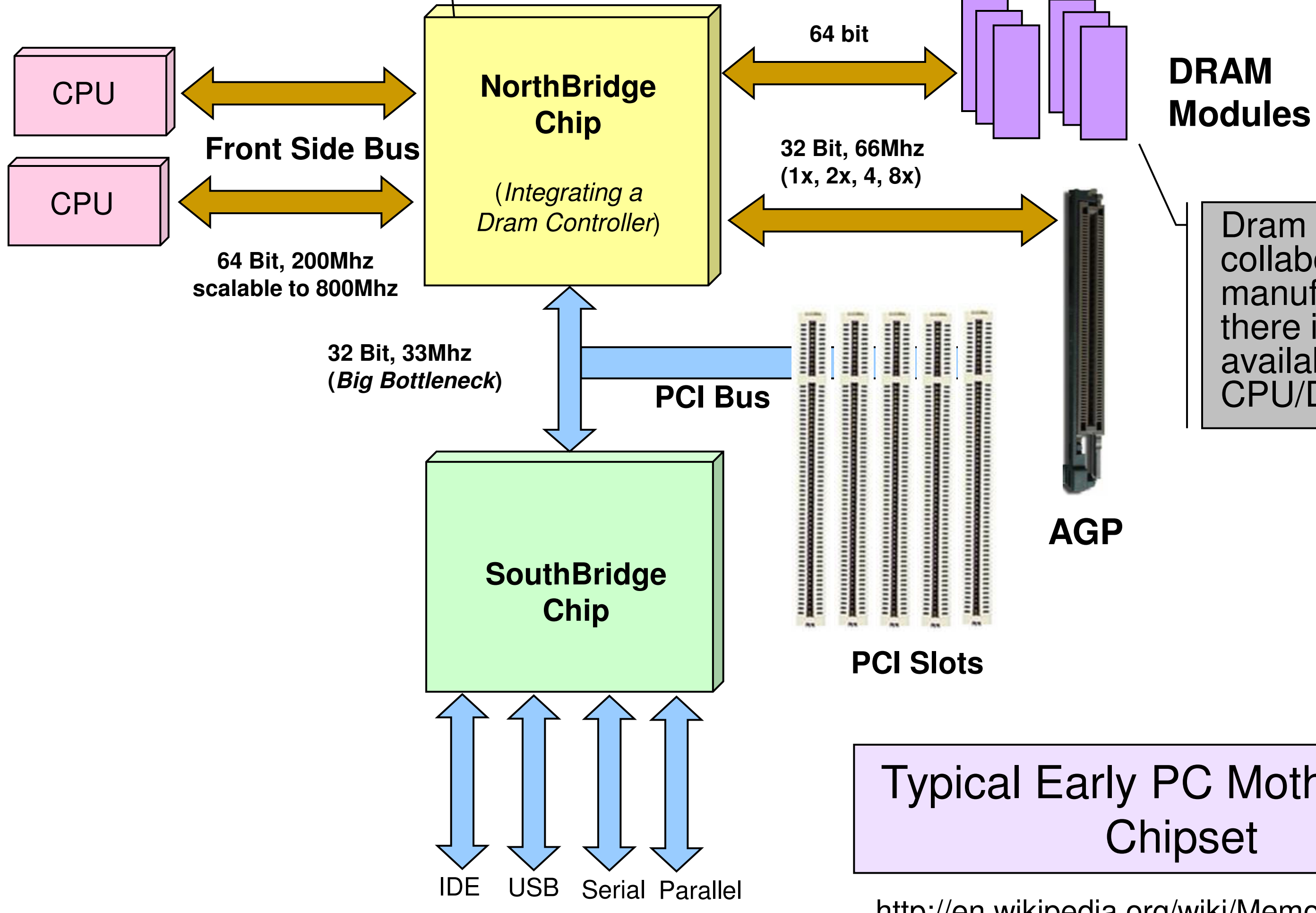
  Additional refresh circuitry has to be built into the dram controller, increasing the cost and complexity of the resulting system.
- **Lower Speed**. Drams are generally slower than their equivalent Srams which is why most high speed computers use caches (made of sram) to store frequently accessed program/data. If you ran your computer directly from dram without a cache it would be pretty slow.

Dram Controllers in Early PCs

Dram technology almost always requires dedicated memory controllers for each combination of host CPU and Dram technology. Can't just hook up CPU directly to the memory chips

Dram controllers handle the complex task of generating the signaling, delicate timing, and refreshing required to get the best performance out of the latest generation of devices.

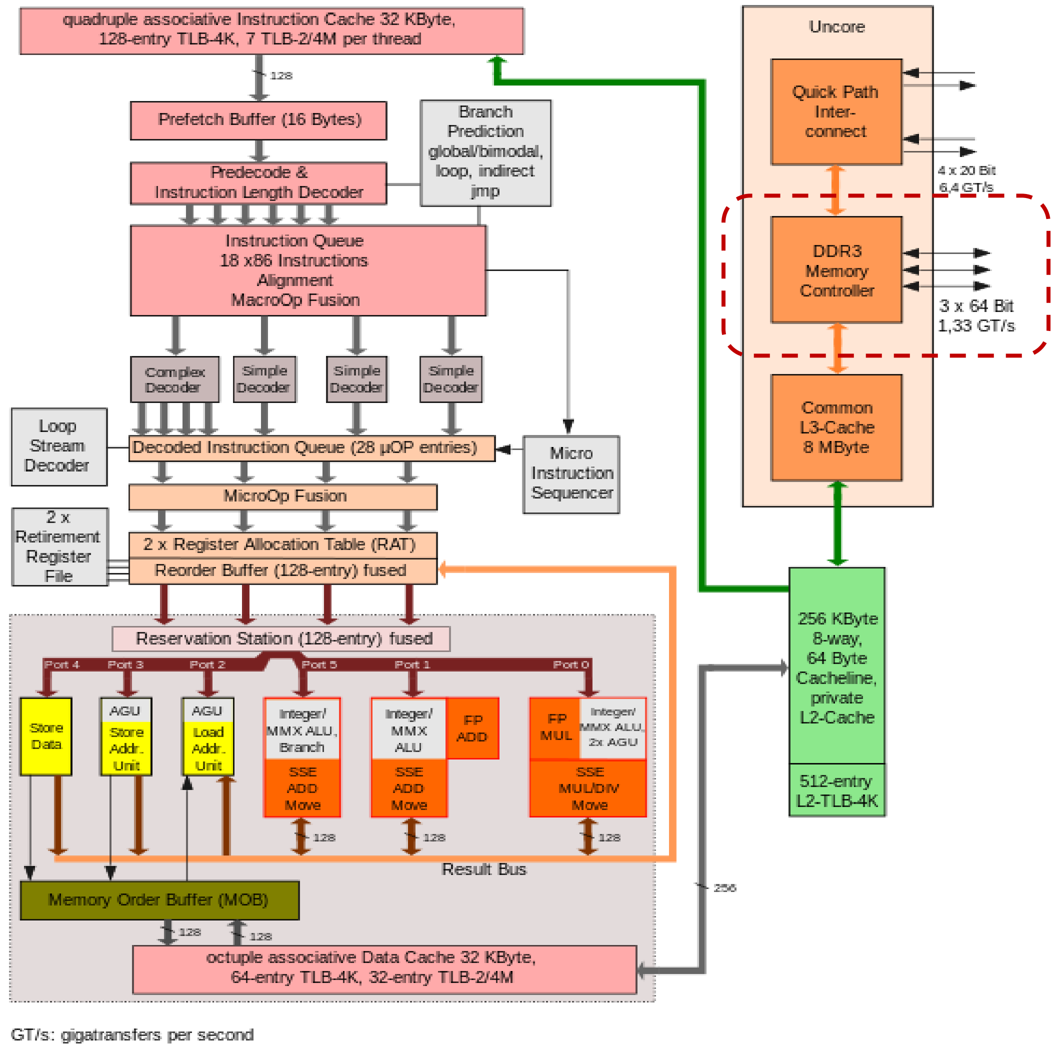
Dram manufacturers collaborate with CPU manufacturers to ensure there is a controller available for the latest CPU/Dram technology


CPU
CPU
Front Side Bus
64 Bit, 200Mhz scalable to 800Mhz
NorthBridge Chip
(*Integrating a Dram Controller*)
64 bit
DRAM Modules
32 Bit, 66Mhz (1x, 2x, 4, 8x)
AGP
32 Bit, 33Mhz (*Big Bottleneck*)
PCI Bus
PCI Slots
SouthBridge Chip
IDE
USB
Serial
Parallel

Typical Early PC Motherboard Chipset

http://en.wikipedia.org/wiki/Memory_controller

Dram Controllers now Integrated into I7 CPU

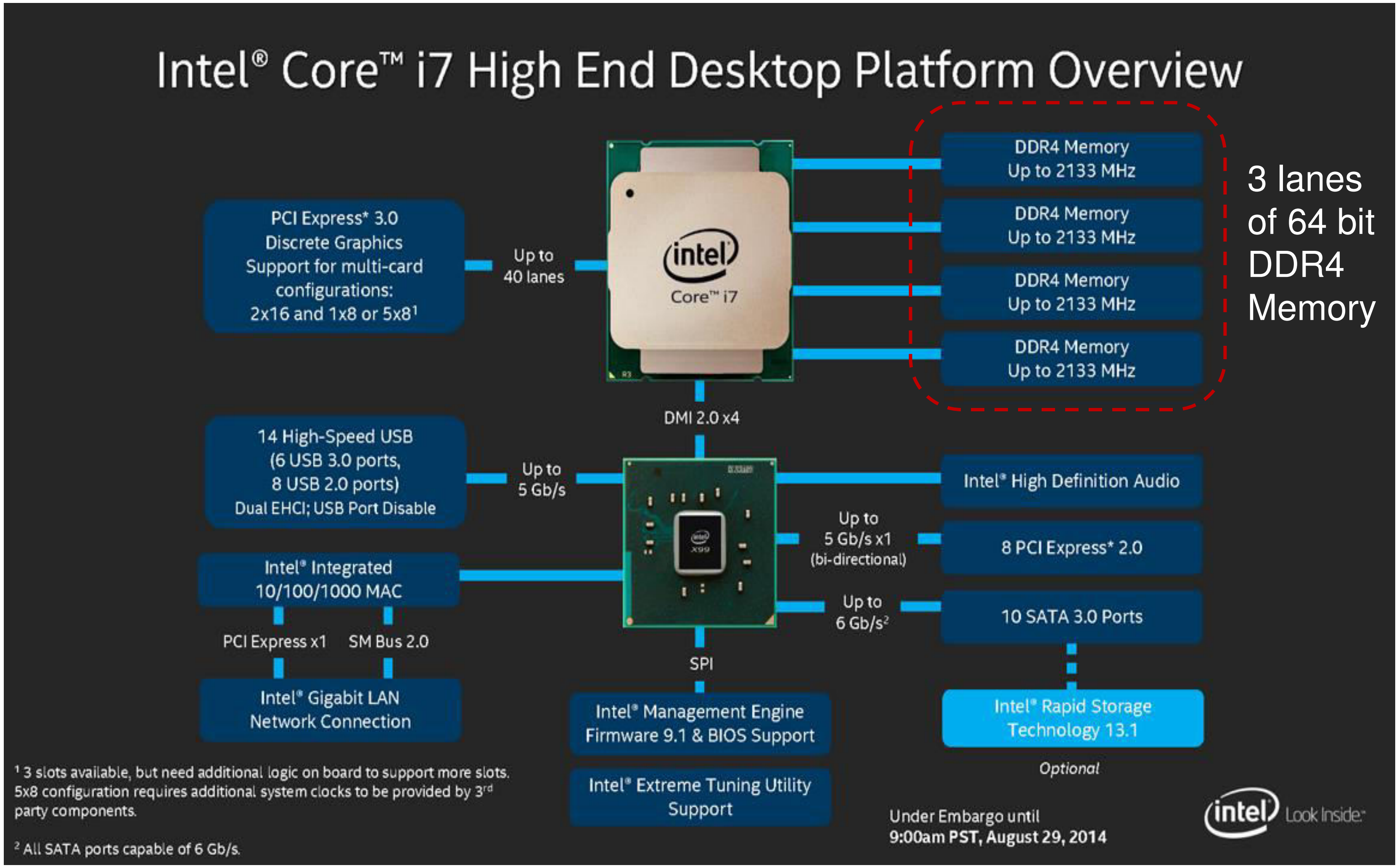

quadruple associative Instruction Cache 32 KByte, 128-entry TLB-4K, 7 TLB-2/4M per thread
128
Prefetch Buffer (16 Bytes)
Branch Prediction global/bimodal, loop, indirect jmp
Predecode & Instruction Length Decoder
Instruction Queue 18 x86 Instructions Alignment MacroOp Fusion
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Loop Stream Decoder
Decoded Instruction Queue (28 µOP entries)
Micro Instruction Sequencer
MicroOp Fusion
2 x Retirement Register File
2 x Register Allocation Table (RAT)
Reorder Buffer (128-entry) fused
Reservation Station (128-entry) fused
Port 4
Port 3
Port 2
Port 5
Port 1
Port 0
Store Data
AGU Store Addr. Unit
AGU Load Addr. Unit
Integer/ MMX ALU, Branch
SSE ADD Move
Integer/ MMX ALU
FP ADD
SSE ADD Move
FP MUL
Integer/ MMX ALU, 2x AGU
SSE MUL/DIV Move
128
Result Bus
Memory Order Buffer (MOB)
octuple associative Data Cache 32 KByte, 64-entry TLB-4K, 32-entry TLB-2/4M
GT/s: gigatransfers per second
Uncore
Quick Path Inter-connect
4 x 20 Bit 6,4 GT/s
DDR3 Memory Controller
3 x 64 Bit 1,33 GT/s
Common L3-Cache 8 MByte
256 KByte 8-way, 64 Byte Cacheline, private L2-Cache
512-entry L2-TLB-4K
256

3 lanes of 64 bit wide memory

Intel I7 “Nehalem” core with integrated DDR3 memory controller

New "Haswell" I7 Processor Motherboard

DDR4 memory controller built into CPU – **Northbridge chipset** is "**no more**"



Intel® Core™ i7 High End Desktop Platform Overview
PCI Express* 3.0 Discrete Graphics Support for multi-card configurations: 2x16 and 1x8 or 5x8¹
Up to 40 lanes
intel Core™ i7
DDR4 Memory Up to 2133 MHz
DDR4 Memory Up to 2133 MHz
DDR4 Memory Up to 2133 MHz
DDR4 Memory Up to 2133 MHz
DMI 2.0 x4
14 High-Speed USB (6 USB 3.0 ports, 8 USB 2.0 ports) Dual EHCI; USB Port Disable
Up to 5 Gb/s
Intel® High Definition Audio
Up to 5 Gb/s x1 (bi-directional)
8 PCI Express* 2.0
Intel® Integrated 10/100/1000 MAC
Up to 6 Gb/s²
10 SATA 3.0 Ports
PCI Express x1
SM Bus 2.0
SPI
Intel® Gigabit LAN Network Connection
Intel® Management Engine Firmware 9.1 & BIOS Support
Intel® Rapid Storage Technology 13.1
Optional
Intel® Extreme Tuning Utility Support
Under Embargo until 9:00am PST, August 29, 2014
intel Look Inside.

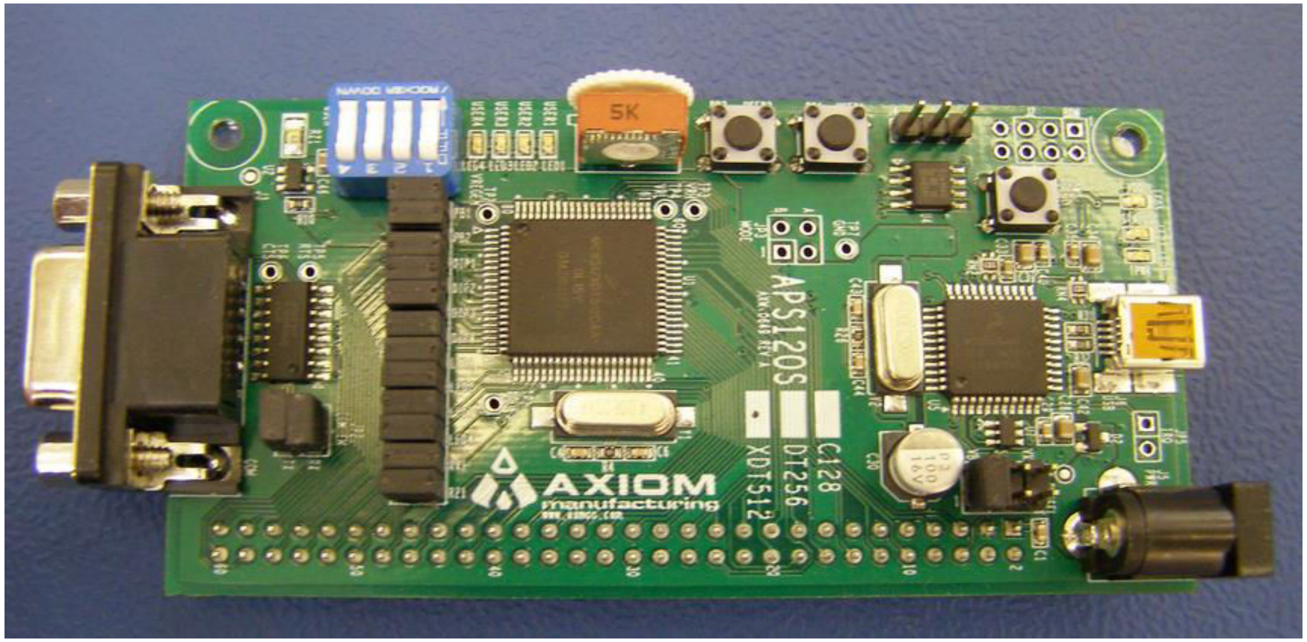
¹ 3 slots available, but need additional logic on board to support more slots. 5x8 configuration requires additional system clocks to be provided by 3rd party components.

² All SATA ports capable of 6 Gb/s.

3 lanes of 64 bit DDR4 Memory

Dram and Microcontrollers

- It is highly unlikely that you would ever use Dram for **8** or even **16** bit systems. The reasons for this are
  - Most microcontroller programs are stored in Flash Eprom, not loaded into static or dynamic memory so the *need* for large amounts of R/W memory is more limited.
  - 8 and 16 bit systems just cannot address large amounts of memory anyway and are aimed at simpler applications.
  - Even relatively large amounts of Sram are inexpensive. For example a 4Meg x 16 SRam interfaces directly to a CPU with minimal complexity.



- 128KB Flash EPROM
- 4KB EEPROM
- 32KB SRAM
- 8-ch, 10-bit, ADC
- 8-bit Timer
- 7-ch, 8-bit PWM
- 9 KBI inputs
- 56 GPIO
- 3 CAN Channels
- 2 SCI & 2 SPI Channels
- I2C Channel

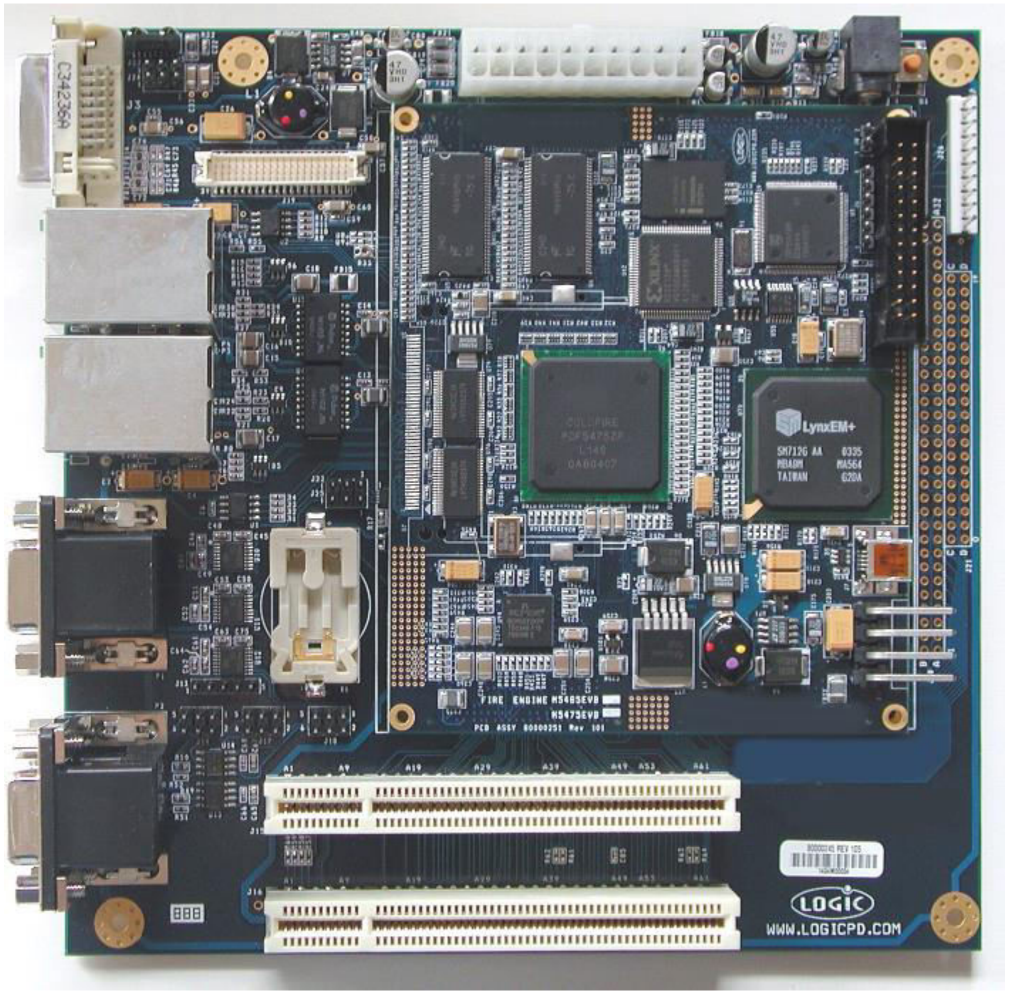
6812 - 16 bit processor board from Axion Manufacturing

Dram and Microcontrollers

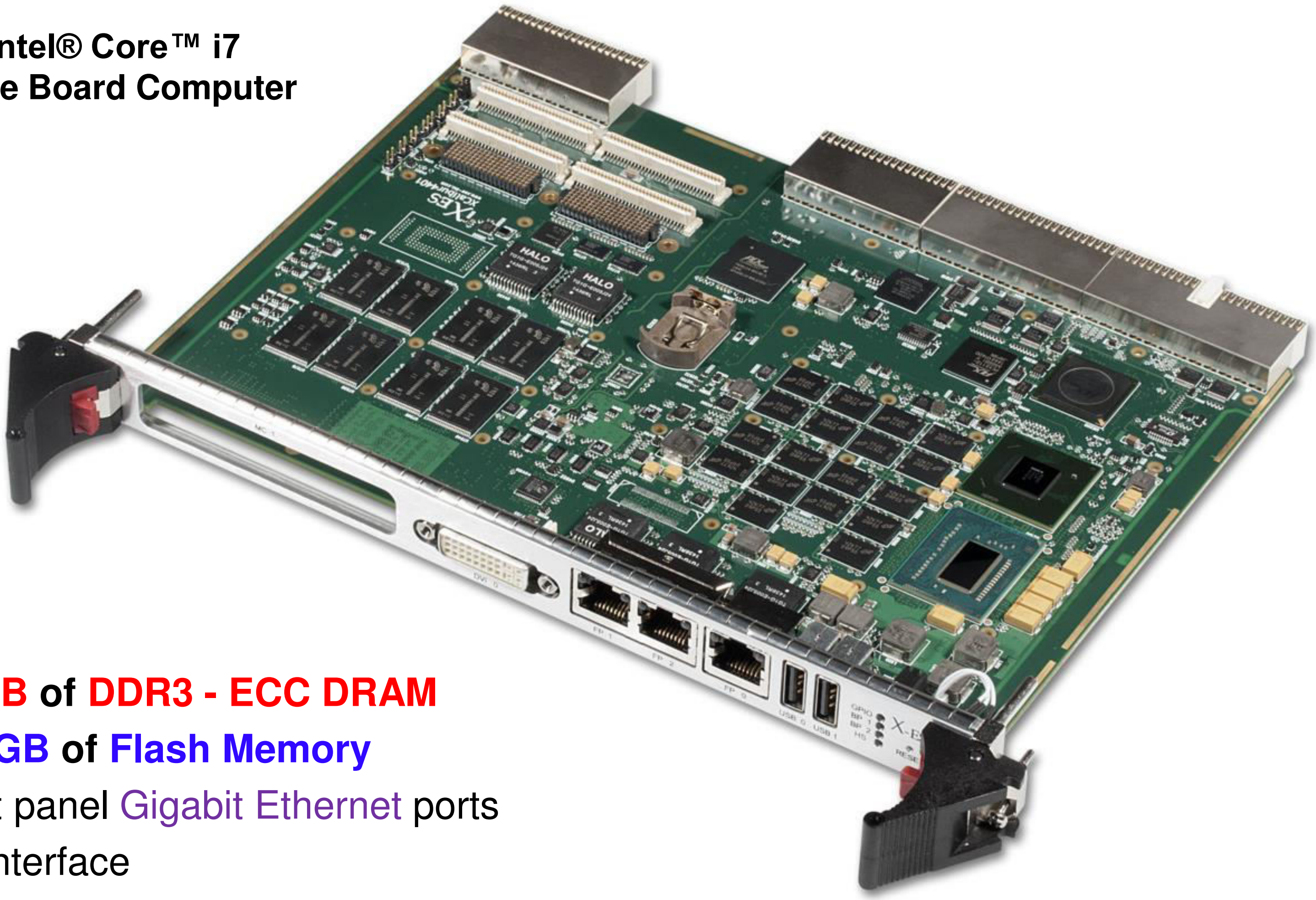
- For more sophisticated 32 and 64 bit processors their larger address space means they run more sophisticated code (e.g. graphics and real time operating systems) making Dram more suitable.

- Today a number of 32 bit microcontrollers have an integrated a Dram controller making it easy to interface with external dram. e.g.

- Opposire is a cold Coldfire (*68k based*) development board from FreeScale with 64MB Dram + Graphics, network etc



XCalibur4401 - Intel® Core™ i7 Processor Single Board Computer



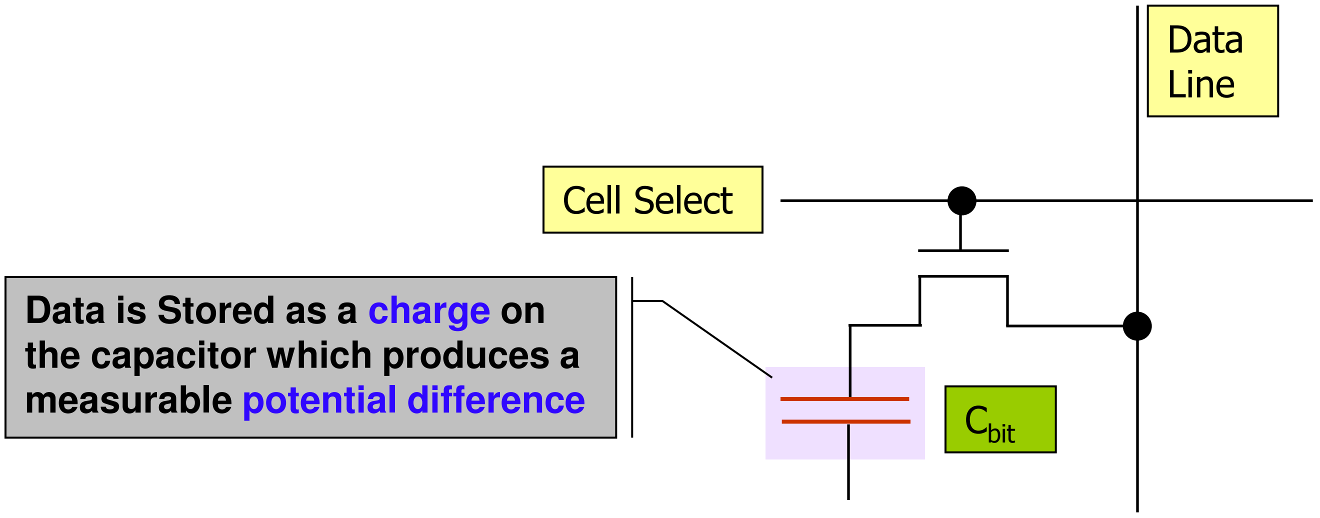
- Up to **16 GB** of **DDR3 - ECC DRAM**
- Up to **128 GB** of **Flash Memory**
- Three front panel Gigabit Ethernet ports
- Graphics Interface
- Two USB 2.0 ports
- Four SATA ports
- Front and rear graphics ports
- Wind River VxWorks, Linux, Microsoft Windows, Neutrino, and LynuxWorks LynxOS operating systems

Example of an Intel I7 based industry processor board with Compact PCi bus interface

Dram Technology: The DRAM Cell

Dram Basics

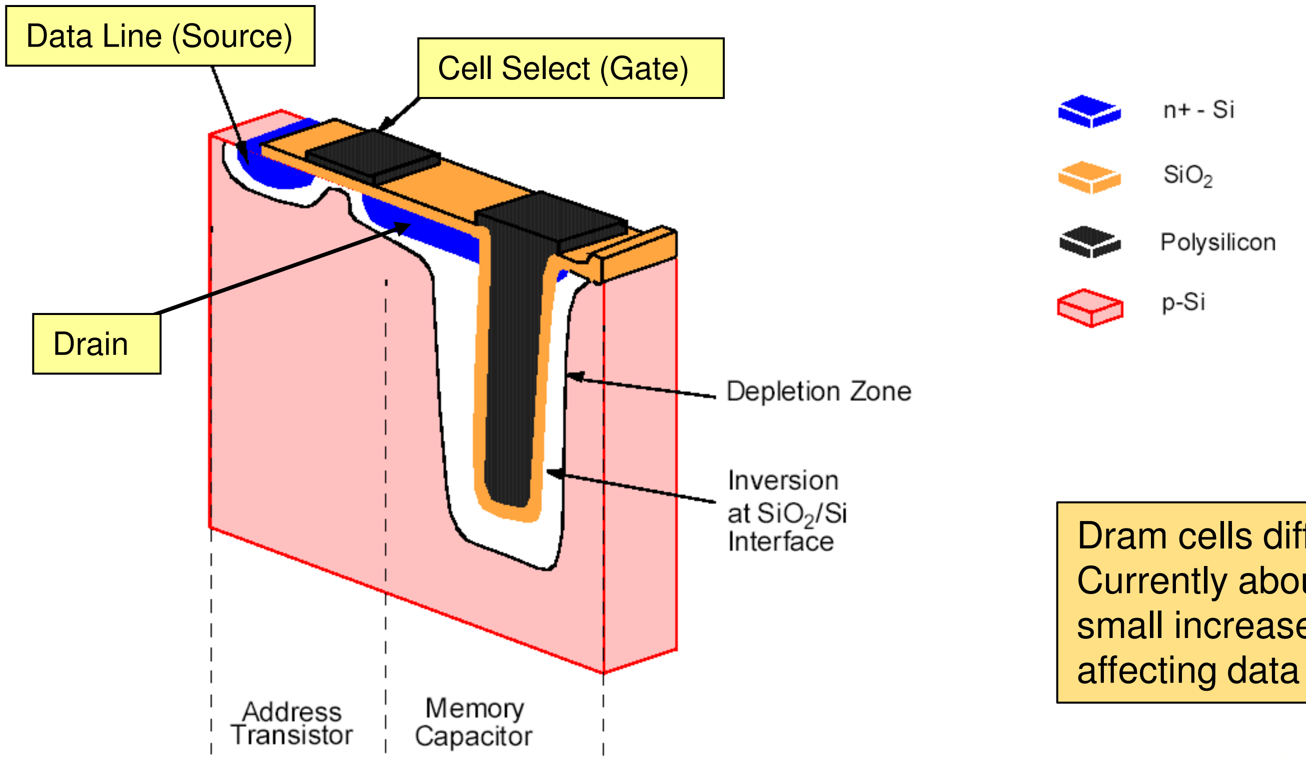
- DRAM uses only one transistor per bit of storage as shown below.
- To understand the operation of this cell we have to look beyond pure *logic* and consider some physics associated with the manufacture of MOSFETs.
- A logic 1 or 0 is maintained, either by the *presence* or *absence* of a charge, across a capacitor C_{bit}. The charge gives rise to a voltage of about 250mV.


Data Line
Cell Select
Data is Stored as a charge on the capacitor which produces a measurable potential difference
C_{bit}
0v

Single Bit Dram Cell

Dram Technology: The DRAM Cell

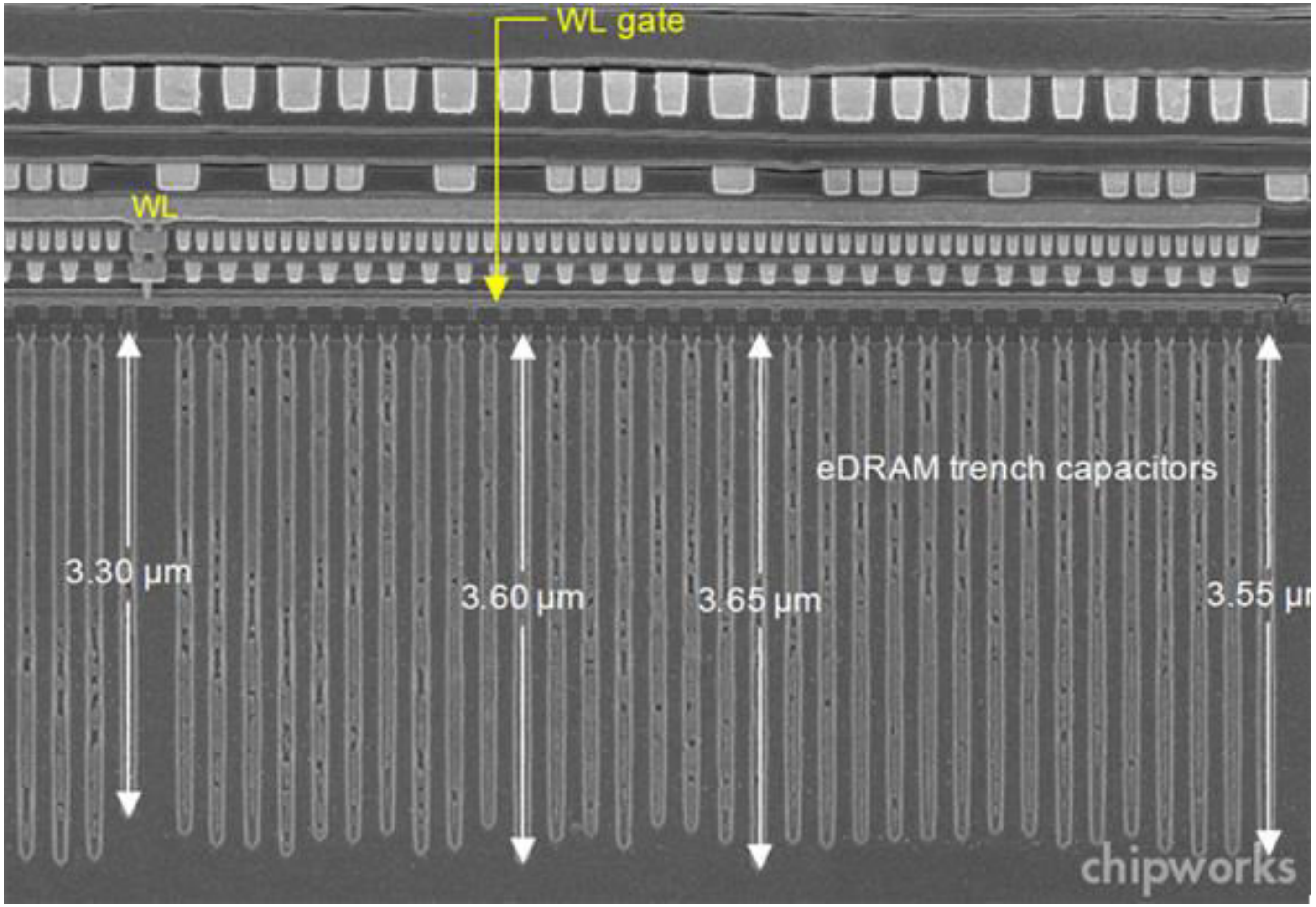
- Several common methods have been used to fabricate the inherent ‘capacitor’.
- Perhaps the earliest is the ‘trench capacitor’ method illustrated below which etches a hole or a ‘well’ into the substrate and connects it to the Drain of the Mosfet.
- The well is lined with Silicon Dioxide and filled with Poly silicon because of its good dielectric properties (*i.e. makes a big capacitor value relative to most other materials*). Even so the capacitor is small in value, typically <10^{-12} farads i.e. <1pf.



Dram cells difficult to scale. Currently about 20nM, going small increases leakage affecting data retention time

Dram Technology: The DRAM Cell

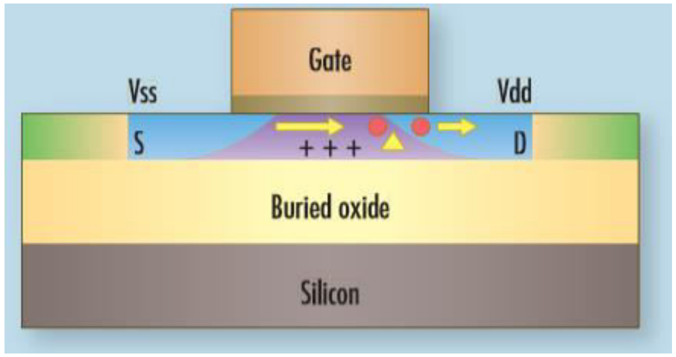
- Microscopic view of Intel’s embedded dram capacitors clearly shown buried in substrate. This was used to make cache memory on latest I7 processor.



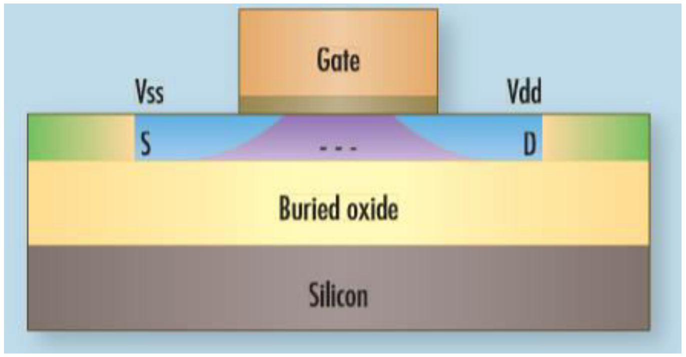
Dram Technology: The DRAM Cell

- More recently, technology has created an inherent capacitor *buried* under the transistor, the so called 'Z-Cell', **1T** or zero capacitor dram cell, which takes up less space and increases the storage density on the chip.

Logic 1



Logic 0

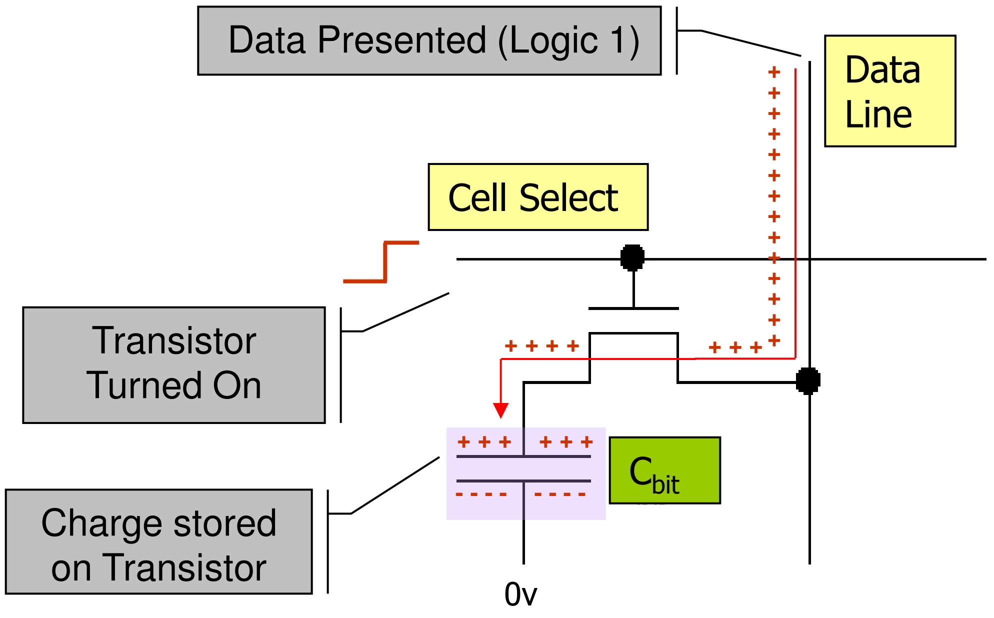


Z-RAM relies on the floating body effect, which was first encountered in the new silicon on insulator (SOI) process introduced in the early 2000s This effect causes capacitance to form between the transistor and the underlying insulating substrate, and was considered a *problem* that needed to be solved in conventional digital systems. The same effect, however, allows a DRAM-like cell to be built using the transistor only, the floating body effect taking the place of the conventional trench capacitor. (http://en.wikipedia.org/wiki/Z-RAM)

Dram Technology: The DRAM Cell

Storing a Logic 1 in a Cell

- A logic 1 is first placed onto the *Data* line.
- The *Cell Select* signal, (*from the chips internal address decoder*), is taken high when an address is presented to the chip. This turns on the transistor.
- Positive charge will flow from the data line and will be deposited onto C_{bit} creating a voltage potential across the capacitor.
- The Cell select line is then negated, leaving the charge trapped on the capacitor.

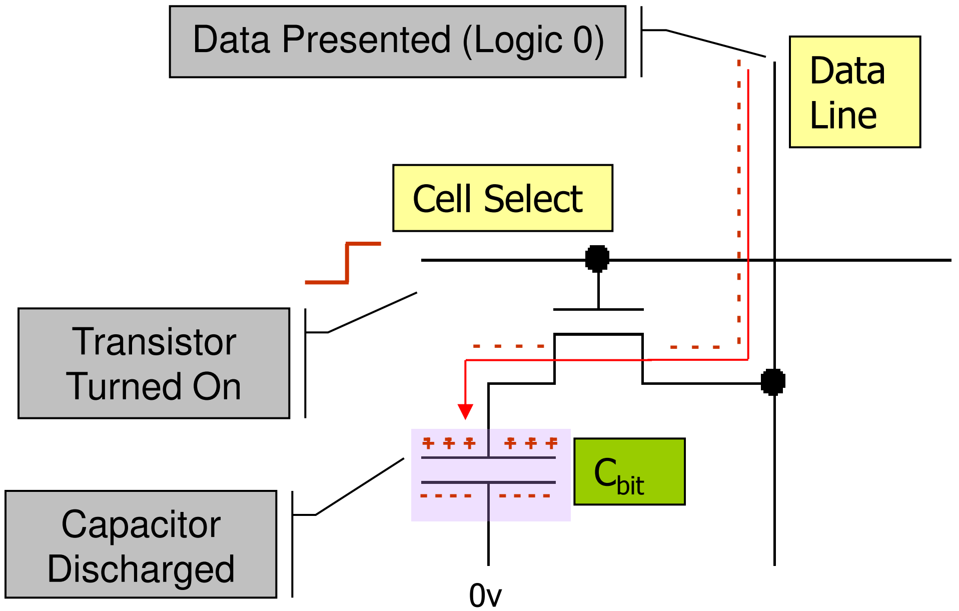


Writing a Logic 1

Dram Technology: The DRAM Cell

Storing a Logic 0 in a Cell

- A logic 0 is placed on the *Data* line
- The *Cell Select* signal is taken high when an address is presented to the chip. This turns on the transistor.
- Any positive charge that might have been stored on the capacitor (*due to it previously storing a logic 1*) will be discharged creating no voltage potential across the capacitor.
- The Cell select line is negated, leaving the capacitor discharged.

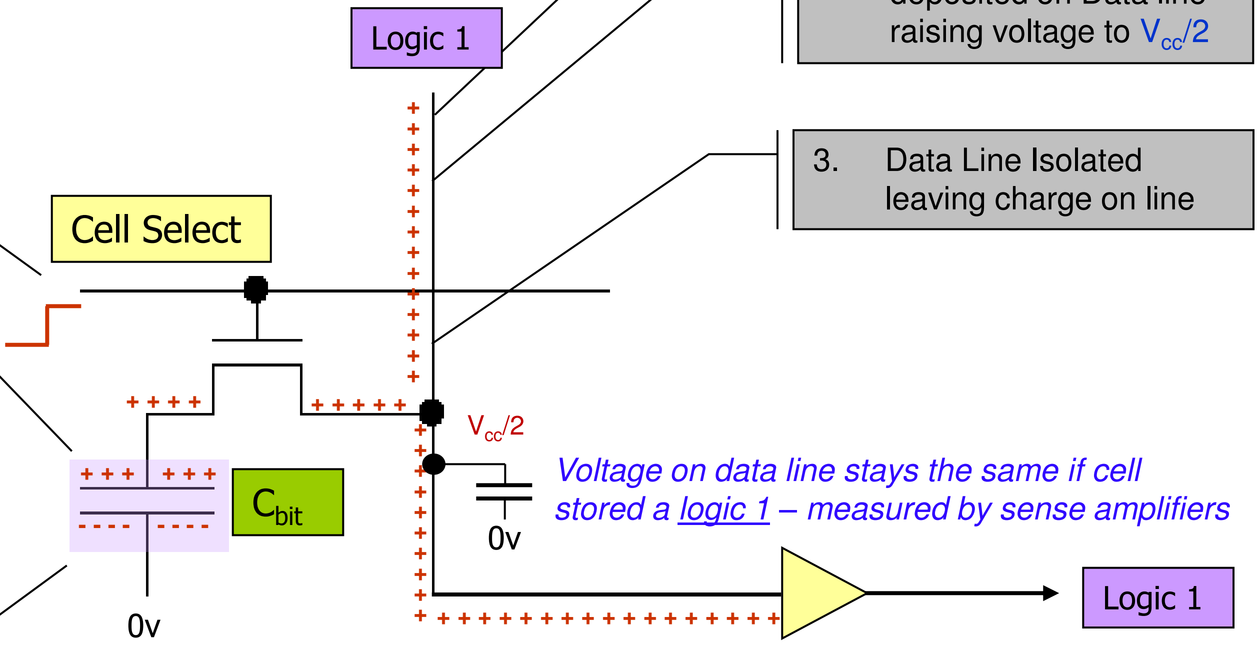


Writing a Logic 0

Dram Technology: The DRAM Cell

Reading from a Cell with a Logic 1 stored

1. Data Line set to Logic 1
2. Minute Charge deposited on Data line raising voltage to $V_{cc}/2$
3. Data Line Isolated leaving charge on line
4. Cell Select Pulsed On, transistor conducts
5. Charge stored on Capacitor equal to charge on Data line so voltage potential on Data Line remains constant



Voltage on data line stays the same if cell stored a logic 1 – measured by sense amplifiers

Cell Storing a Logic 1

Sensitive Charge Amplifier on chip

6. Sense Amplifier sees enough charge on Data Line to register a logic 1

Dram Technology: The DRAM Cell

Reading from a Cell with a Logic 0 stored

1. Data Line set to Logic 1

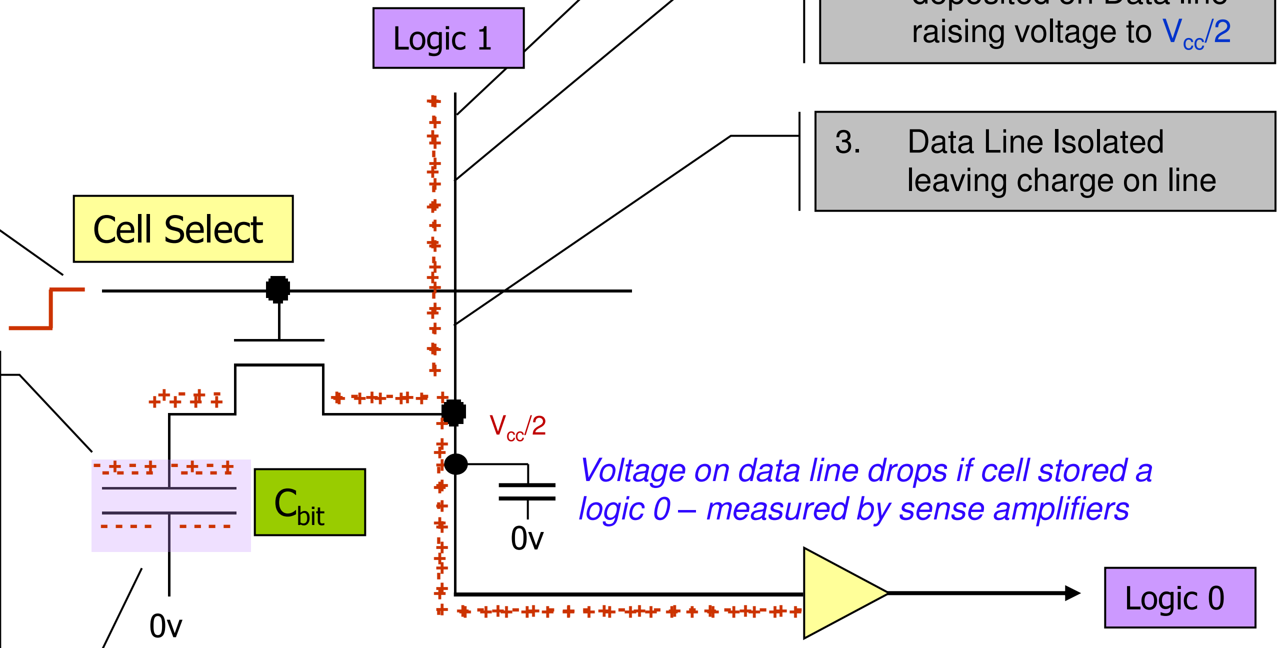
2. Minute Charge deposited on Data line raising voltage to $V_{cc}/2$

3. Data Line Isolated leaving charge on line

4. Cell Select Pulsed On, Transistor conducts

5. Charge flows from Data line onto capacitor reducing the charge and hence the voltage on the data line.

   Reading (a logic 0) destroys the charge and hence the data stored in the Cell so it has to be re-written afterwards



Voltage on data line drops if cell stored a logic 0 – measured by sense amplifiers

Sensitive Charge Amplifier on chip

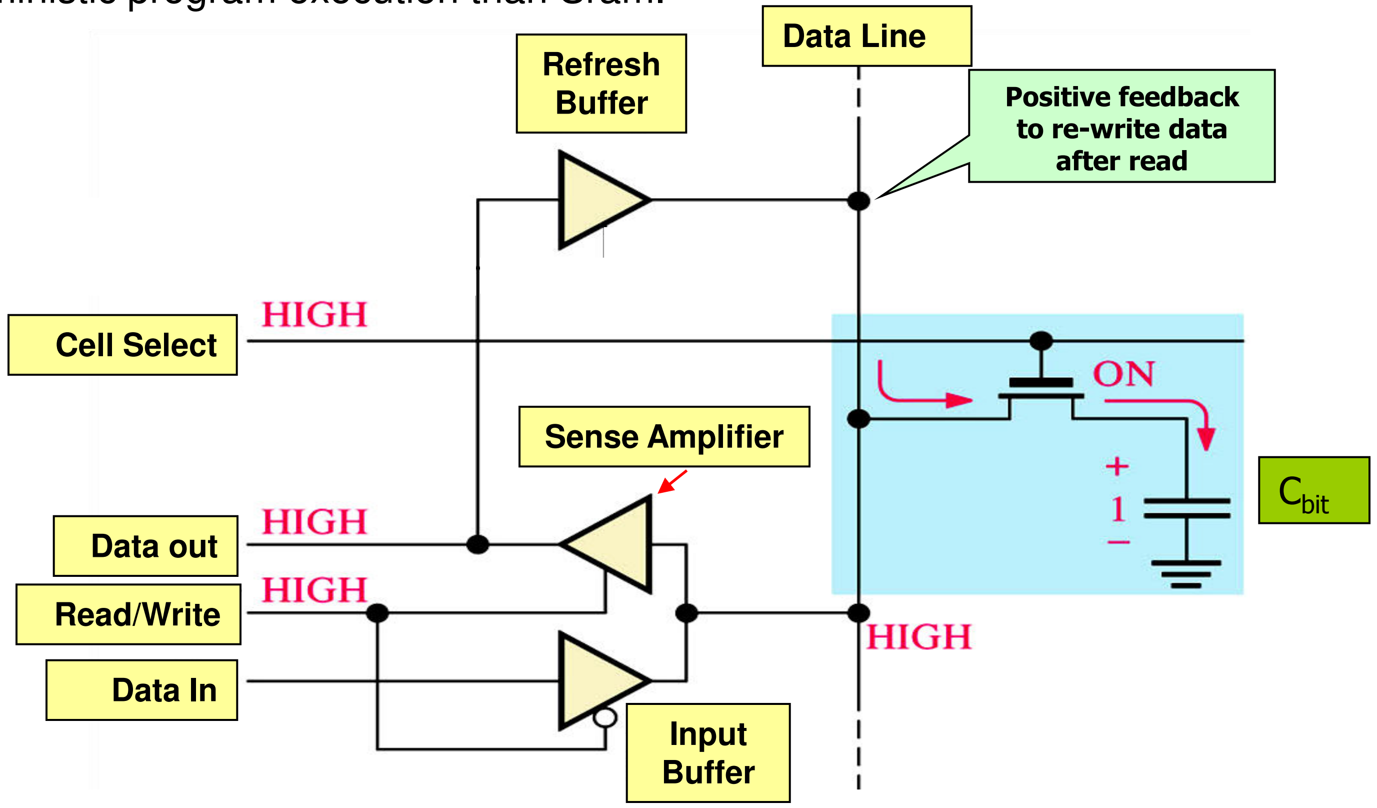
Cell Storing a Logic 0

6. Sense Amplifier does not see enough charge on data line to register a logic 1 so registers a logic 0 instead

Problems with Dram Technology

Charge Leakage

- The minute charge stored on (C_{bit}), tends to leak away with time leading to lost data and is also corrupted by the action of reading the cell.
- Refreshing must therefore be performed whenever a cell is read (*see feedback loop below reinforcing the charge stored on the capacitor*), and, in the absence of frequent reads, has to be performed periodically by external hardware.
- During a periodic refresh the CPU will not be able to access the chip making for less deterministic program execution than Sram.



Problems with Dram Technology

Problems with Dram Cells: Read Cycle Times

- Reading is a relatively slow process because it requires that a precise measured charge be deposited on the Data line (*so that the sense amplifiers can detect the different between a 0 and 1 stored in the cell – see page 14/15*) which takes time for the chip to generate internally.
- Thus there is a period of time after each access, which the Dram chip needs to recover and thus it cannot be accessed straight away, slowing down subsequent each access.
- This recovery time is know as the Row pre-charge time and is comparable to that of the access time of the device.
- As a consequence of this recovery time, the chips *cycle time* (i.e. the time between *successive CPU accesses* to the memory chip) is about twice as long as the *access time* and thus wait states may be required in very fast systems.

Problems with Dram Technology

Corruption of Data

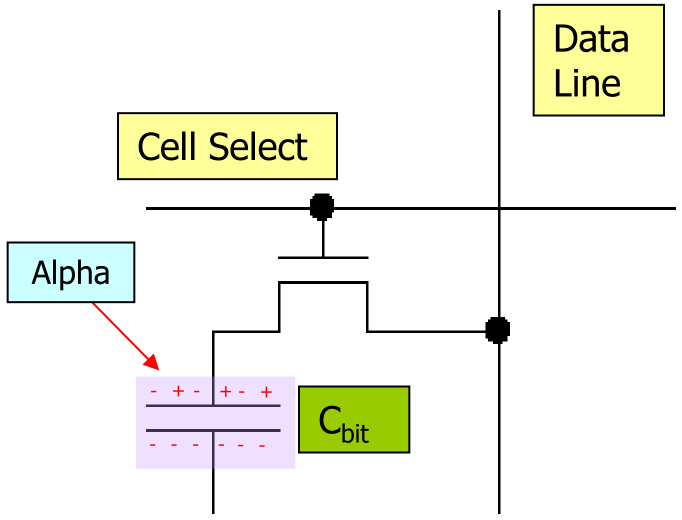
- Dram cells can be corrupted by stray Cosmic particles passing through the device which can cause enough *ionisation* to corrupt the charge stored on the capacitor.

  - This corruption is referred to as a soft-error because the damage to the cell is not permanent but leads to a 1 being incorrectly read as a 0.

  - In high integrity (e.g. military) or high availability (e.g. server) systems, this can be a real problem since the likelihood of soft errors arising is proportional to time.

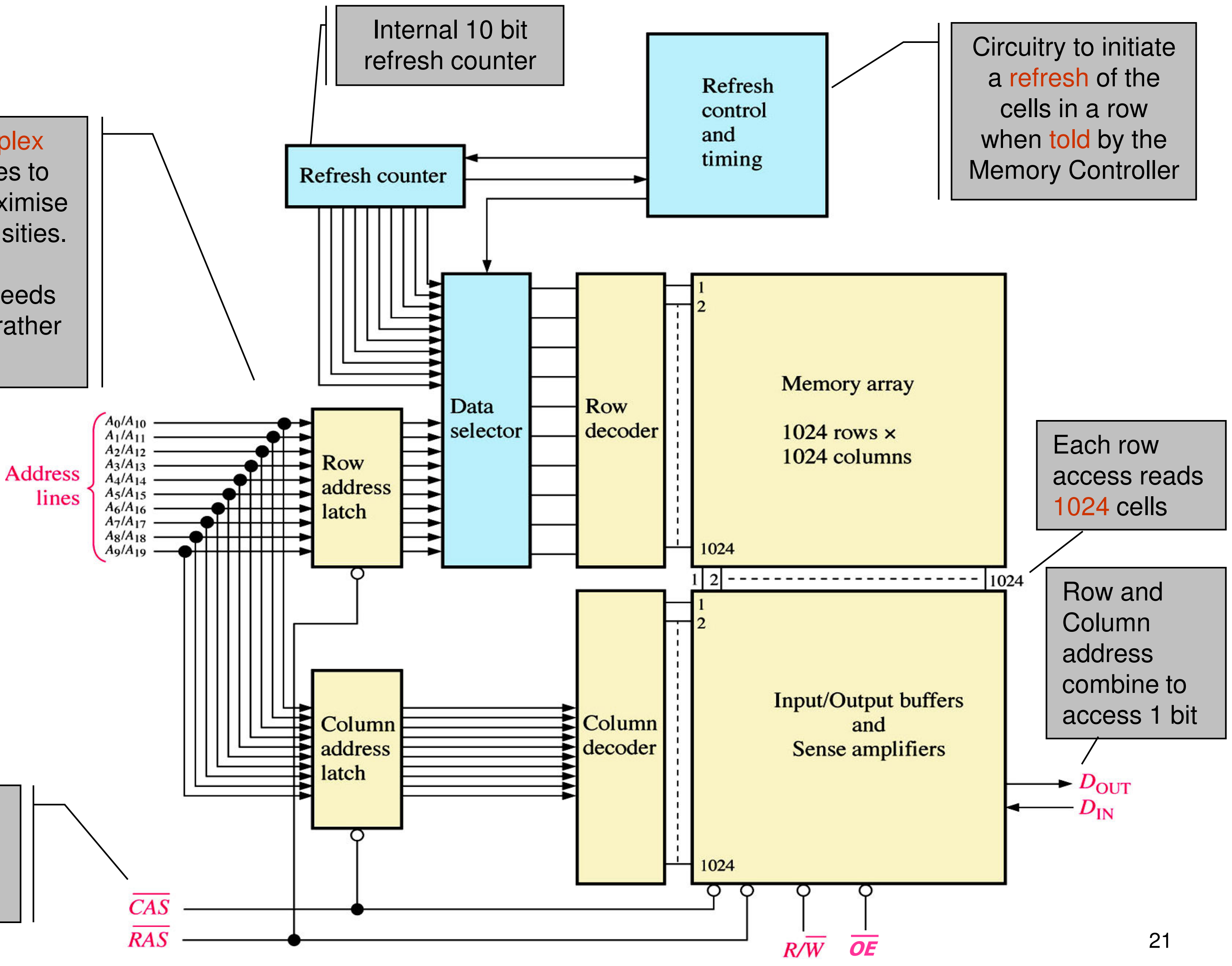
  - In such situations, additional external circuitry may be required to detect and correct the data.

  - More recently special ECC memory chips have become available. These however are much more expensive (*see later lecture*).



Cosmic particles can destroy the charge stored on the C_{bit} corrupting the data stored there

- First generation Drams were '*Asynchronous*' devices with no clocked circuitry. An example architecture of a **1M x 1bit** device is given below.

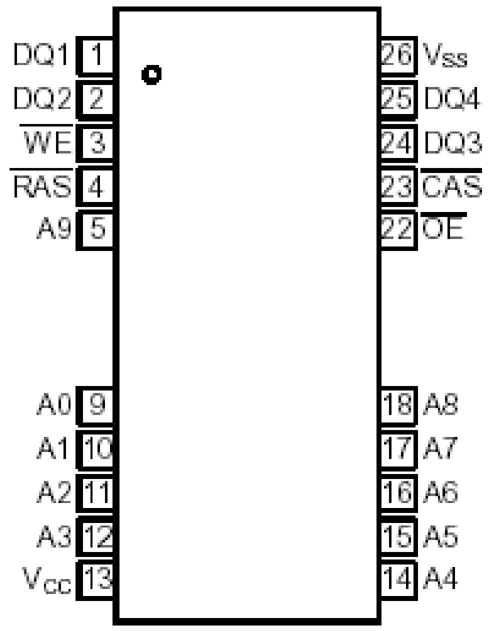


Internal 10 bit refresh counter
Circuitry to initiate a refresh of the cells in a row when told by the Memory Controller
Refresh control and timing
Refresh counter
All Dram's multiplex their address lines to save pins and maximise PCB package densities.
1Meg chip only needs 10 address lines rather than 20
Address lines
A_0/A_{10}
A_1/A_{11}
A_2/A_{12}
A_3/A_{13}
A_4/A_{14}
A_5/A_{15}
A_6/A_{16}
A_7/A_{17}
A_8/A_{18}
A_9/A_{19}
Row address latch
Data selector
Row decoder
Memory array
1024 rows × 1024 columns
1
2
1024
Each row access reads 1024 cells
Row and Column address combine to access 1 bit
Column address latch
Column decoder
Input/Output buffers and Sense amplifiers
D_{OUT}
D_{IN}
Row and Column address Strobes to de-multiplex address bus
$\overline{CAS}$
$\overline{RAS}$
$R/\overline{W}$
$\overline{OE}$

Example Asynchronous Dram (1M x 4 bit Device)

OKI 514400

- Single 5 volt supply
- 1Meg x 4bit data
- 10 Address Lines
- 4 Bi-directional Data Lines
- 60ns Access Time
- Internal Refresh Counter
- Refreshing required for all 1024 rows every 128ms.



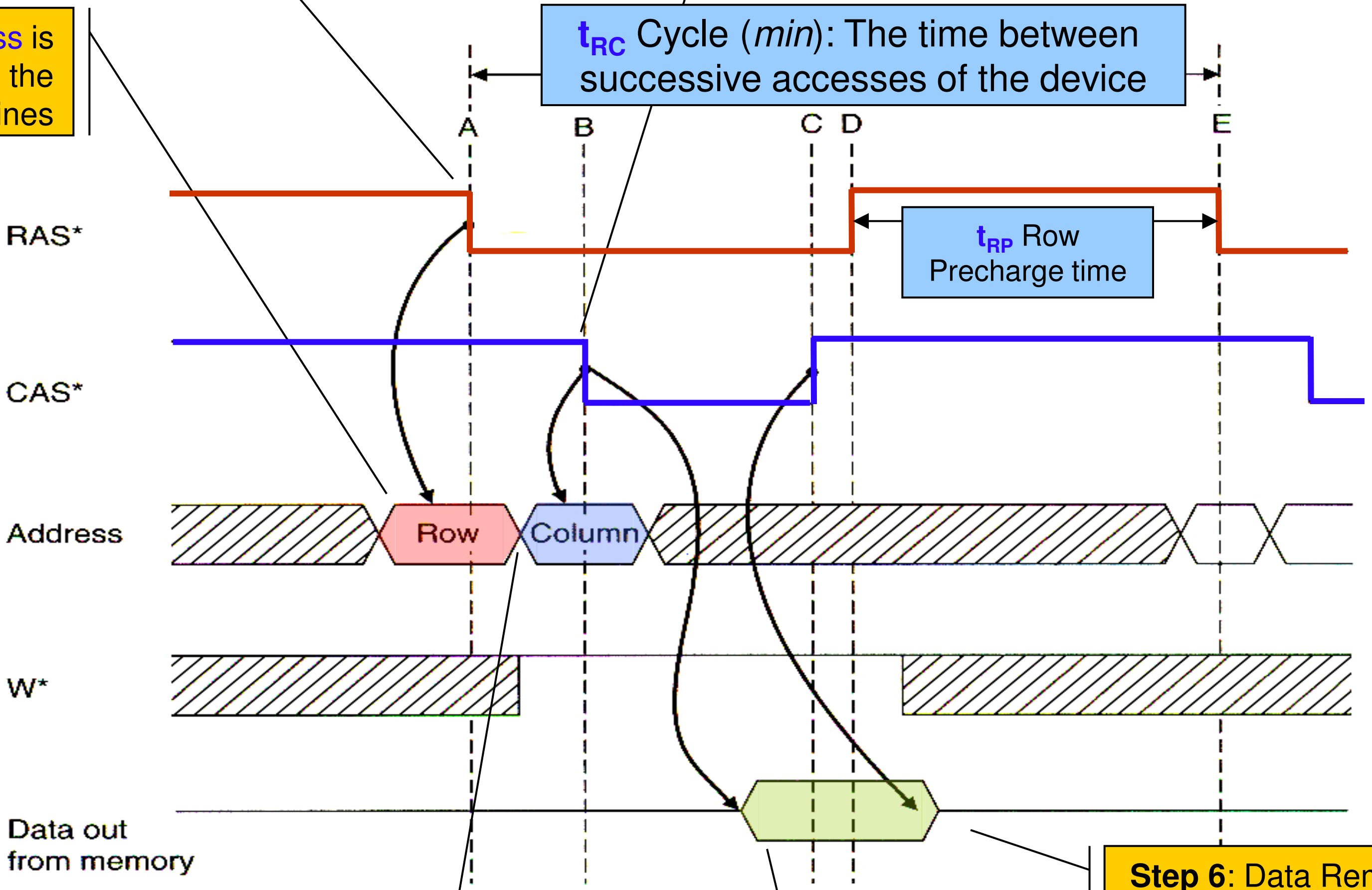
Pin Name	Function
A0–A9	Address Input
$\overline{RAS}$	Row Address Strobe
$\overline{CAS}$	Column Address Strobe
DQ1–DQ4	Data Input/Data Output
$\overline{OE}$	Output Enable
$\overline{WE}$	Write Enable
V_{CC}	Power Supply (5 V)
V_{SS}	Ground (0 V)

A Simplified Asynchronous Dram Read Cycle

Step 2: The RAS* signal is taken low. The device latches the row address internally

Step 1: A row address is multiplexed onto the Dram address lines

Step 4: The CAS* signal is taken low. The device latches the column address internally



t_{RC} Cycle (*min*): The time between successive accesses of the device
A
B
C
D
E
RAS*
t_{RP} Row Precharge time
CAS*
Address
Row
Column
W*
Data out from memory

Step 6: Data Remains valid until Negation of CAS*

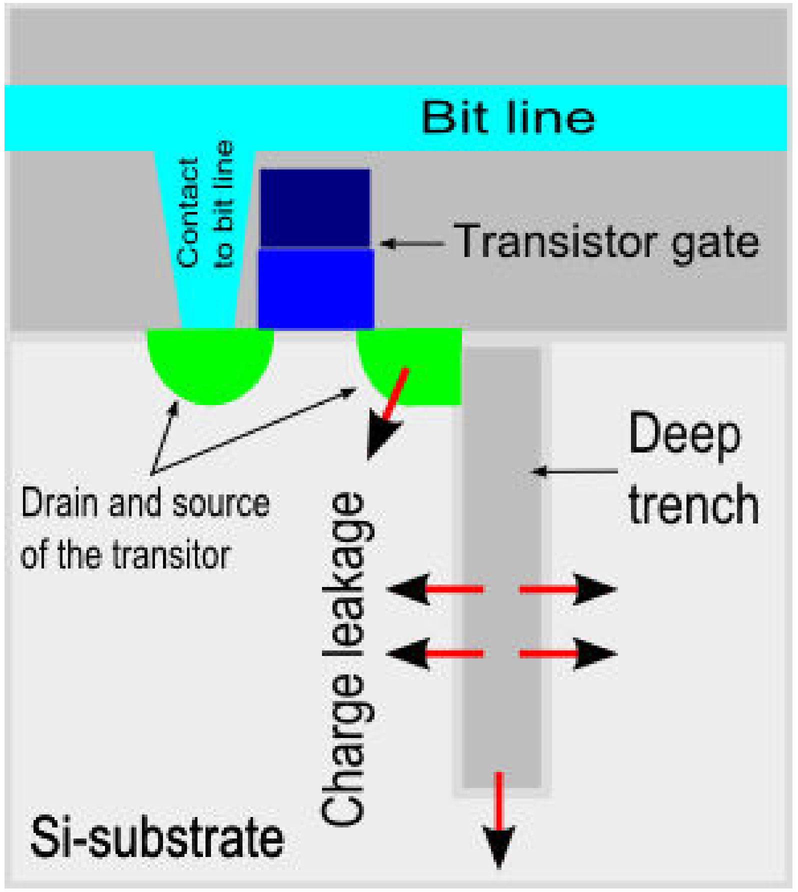
Step 3: External multiplexer must switch over and present the other half of the address, the column address to the Dram

Step 5: Data becomes available **60nS** after RAS or **15ns** after CAS (whichever is slower)

Dram Refreshing

What, Why and How

- Refreshing arises because the charge on the capacitor of the dram cell leaks away.
- Refreshing 'tops-up' the charge before the data is lost due to leakage.
- Few memory chips refresh completely by themselves. External hardware is usually required if only to trigger the device into performing the refresh at the correct time.
- Refreshing involves reading every cell in the chip (*which implicitly means that the charge on the cell gets topped up*) within a period specified by the manufacturer, (*typically 64mS*)



Dram Refreshing

- A single refresh operation will refresh all cells sharing the same row address, (1024 of them in a 1Meg chip) thus we do **not** have to refresh all cells *individually*.

- For a 1Mbit chip such as the OKI 514400 comprising 1024 rows, this means that every row has to be refreshed at least once during the specified refresh period.

- Most modern Dram chips simplify the task of refreshing by having a refresh counter fabricated onto the chip to keep track of which Row is to be refreshed next (see page 19).

- During a refresh, the CPU must not be permitted to access the memory chip.

Dram Refreshing

- Most Drams can be physically refreshed in variety of ways, depending upon the complexity of the controller you wish to build and the dram chip itself.

Refreshing by Reading :-

As we saw in a previous lecture, reading a cell automatically refreshes not only *that* cell, but also all other cells sharing the same Row address.

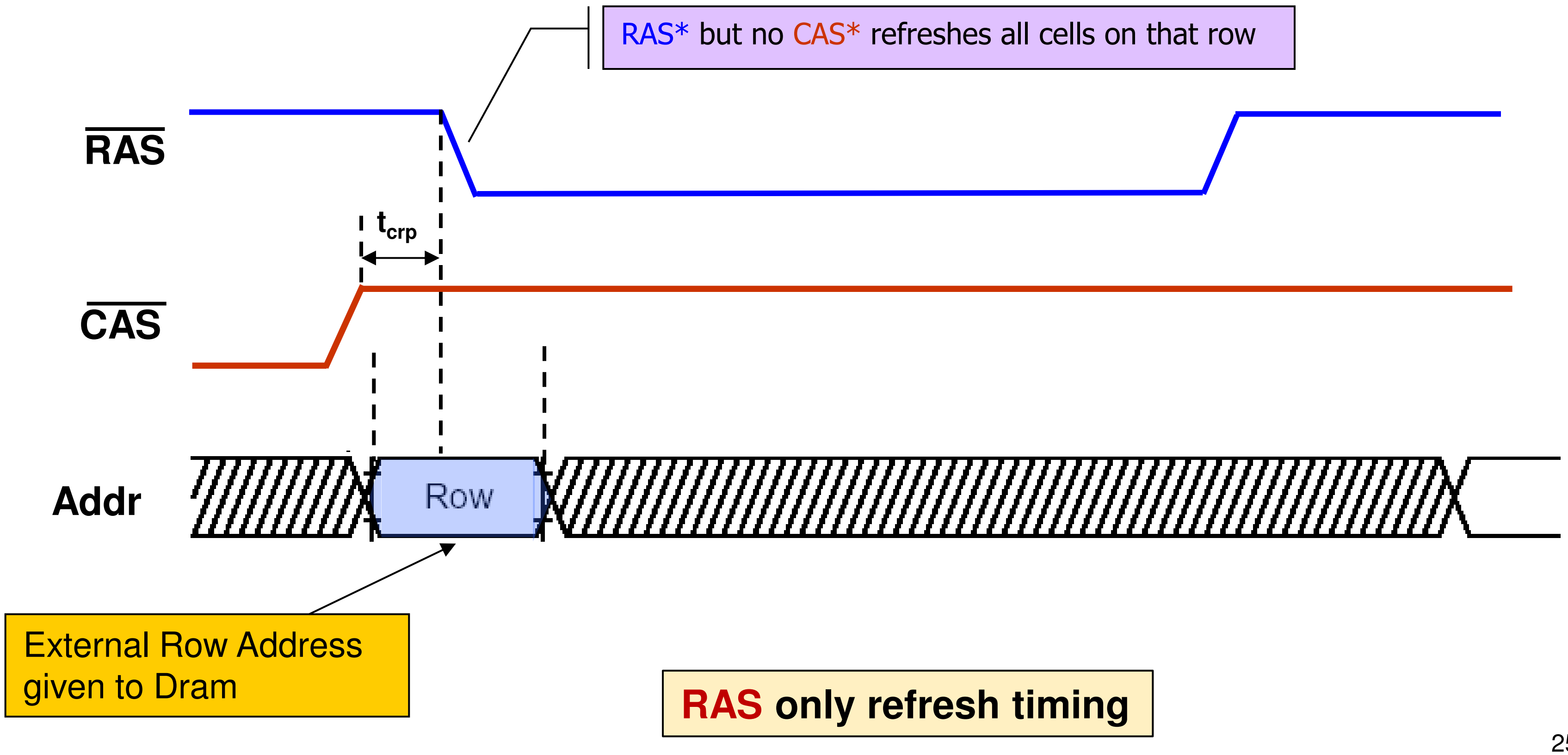
Provided the CPU, during the execution of its program, accesses all row addresses within the chip within the given refresh period of say 128ms, then all cells in the chip will be refreshed automatically.

The flaw in this approach is that there is no guarantee, given the way programs jump around within loops etc. that all row addresses will be issued within the refresh period so it's not a reliable method (*unless you do it as part of an Interrupt service routine designed to refresh this way*) but it work well in graphics cards – can you think why?

Dram Refreshing

RAS* Only Refreshing

- The very 1st generation of Drams used Ras* only refreshing. Here a row address is presented externally to the chip and Ras* is asserted, without the Cas*. This is what distinguishes a refresh from a normal access.
- The row address is typically maintained via an external counter, which is multiplexed onto the dram address lines when a refresh operation is initiated, and incremented afterwards, i.e. more external circuitry.



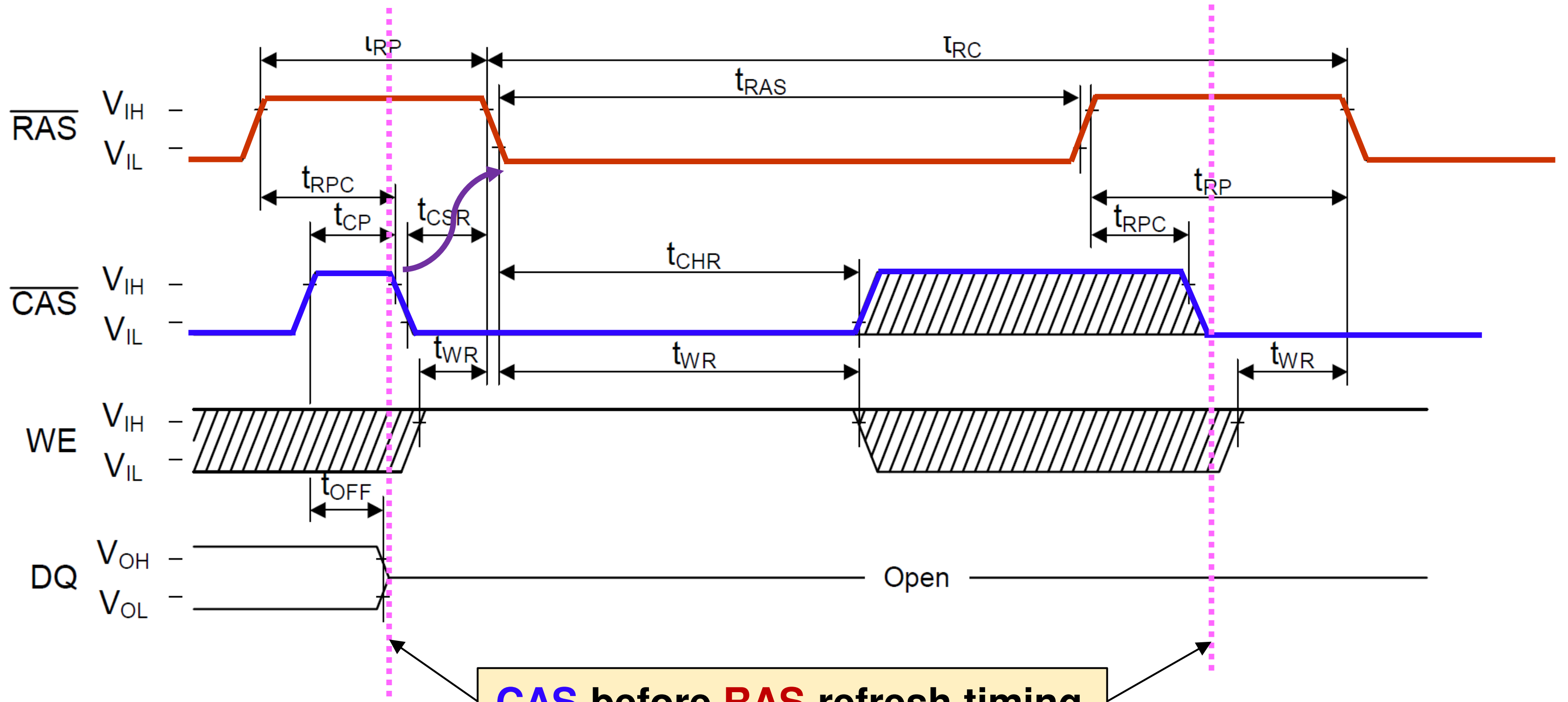
RAS* but no CAS* refreshes all cells on that row
RAS
t_{crp}
CAS
Addr
Row
External Row Address given to Dram

RAS only refresh timing

Dram Refreshing

CAS* before RAS* Refreshing

- Later 1st Generation devices (such as the OKI 514400) had the row counter and multiplexer integrated onto the data chip (reducing external chip count).
- A new refresh protocol was introduced called, **CAS *before* RAS** (*the opposite of a normal access*).
- The internal row counter is auto incremented afterwards.

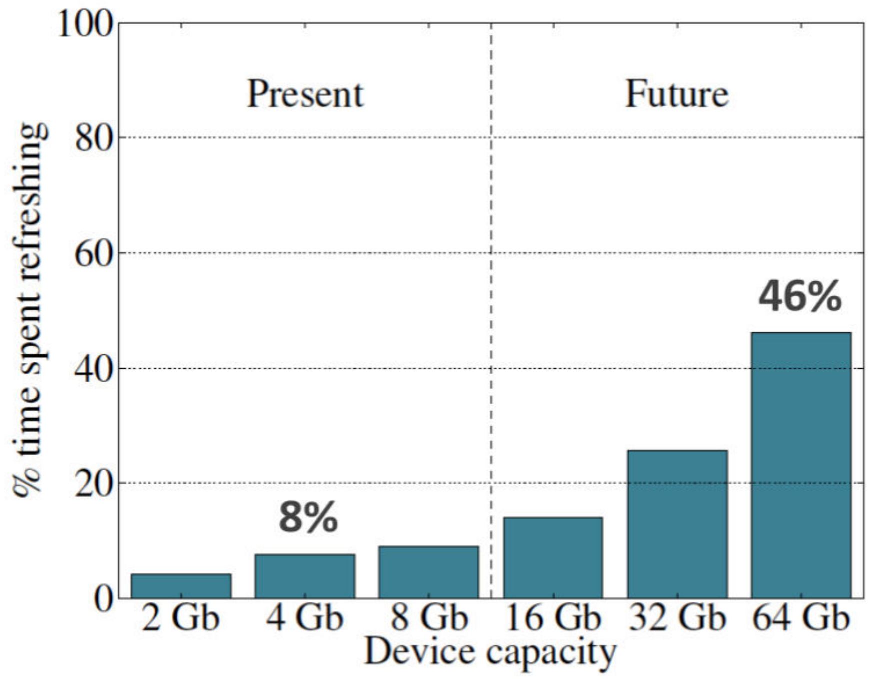


CAS before RAS refresh timing

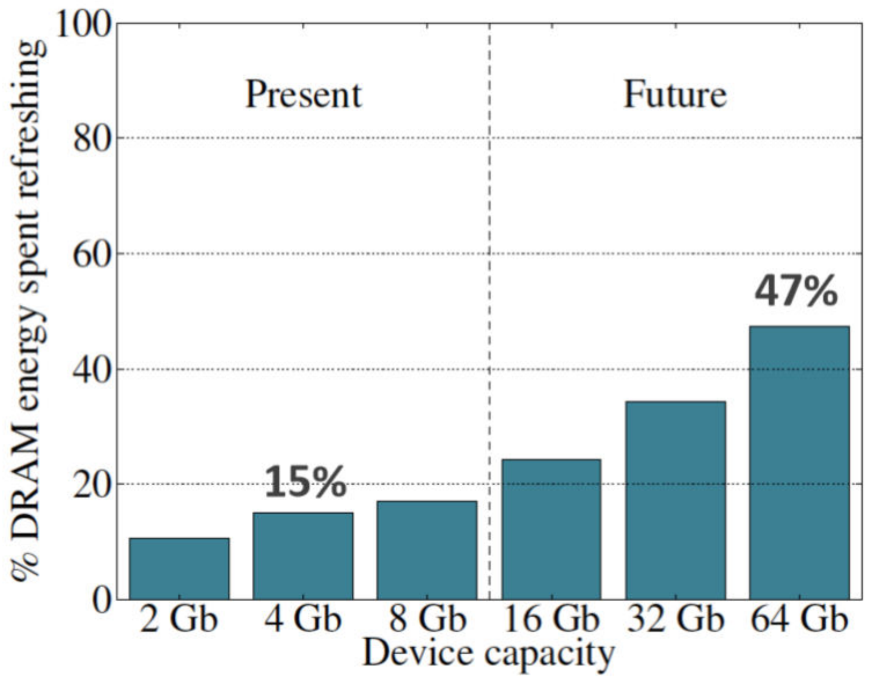
Dram Refreshing

Downsides of Refreshing

- Energy consumption: Each refresh consumes energy.
- Performance degradation: DRAM rank/bank unavailable while refreshed.
- QoS/predictability impact : (Long) pause times during refresh.
- Refresh rate limits DRAM capacity scaling (see graph below).



Performance Impact due to Refreshing



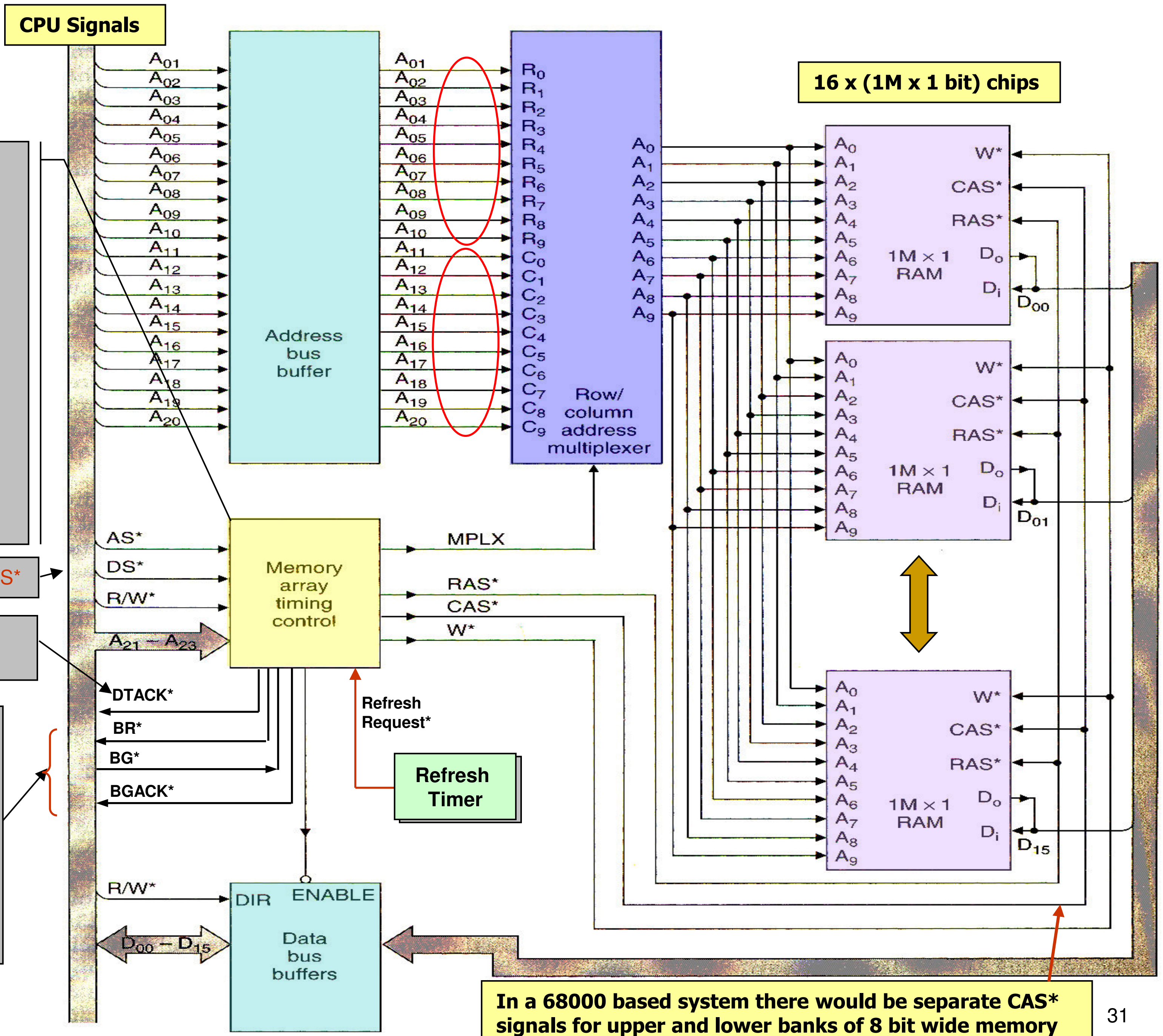
Energy Impact due to Refreshing

Dram Refreshing

- **Observation**: Most DRAM rows can be refreshed much less often without losing data.

- **Key idea**: Refresh rows containing weak cells more frequently, other rows less frequently. This can be done through **Profiling**, i.e. test retention time of all rows and use this data in conjunction with an intelligent dram controller that only refreshes rows when needed.

- **Tricky** as retention time of drams changes with **time/temperature**, may need to profile for many days/weeks to find optimum refresh rate.


CPU Signals
A01 A02 A03 A04 A05 A06 A07 A08 A09 A10 A11 A12 A13 A14 A15 A16 A17 A18 A19 A20
Address bus buffer
R0 R1 R2 R3 R4 R5 R6 R7 R8 R9
C0 C1 C2 C3 C4 C5 C6 C7 C8 C9
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9
Row/ column address multiplexer
16 x (1M x 1 bit) chips
1M × 1 RAM
W* CAS* RAS* Do Di
D00 D01 D15
AS* DS* R/W*
A21 – A23
Memory array timing control
MPLX RAS* CAS* W*
DTACK* BR* BG* BGACK*
Refresh Request*
Refresh Timer
R/W*
DIR ENABLE
Data bus buffers
D00 – D15

Circuit has to
- Decode Address

- Generate RAS/MPLX/CAS timing for a normal CPU Bus Cycle and provide DTACK back to the CPU.

- Initiate periodic Refresh operations

- Take control of the bus during a refresh

DS* means LDS*/UDS*

DTACK* can control wait state generation

Bus request/bus grant Signalling to deal with refreshing

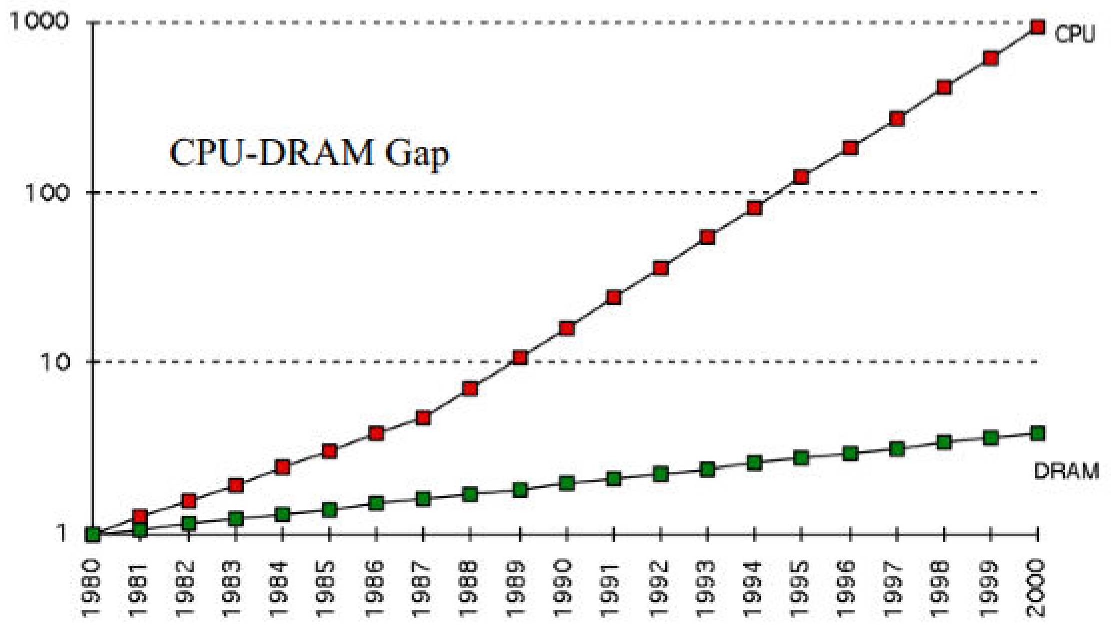
Takes control of the Address and Data bus from the CPU while a refresh operation is performed

In a 68000 based system there would be separate CAS* signals for upper and lower banks of 8 bit wide memory

Dram Vs CPU Speed

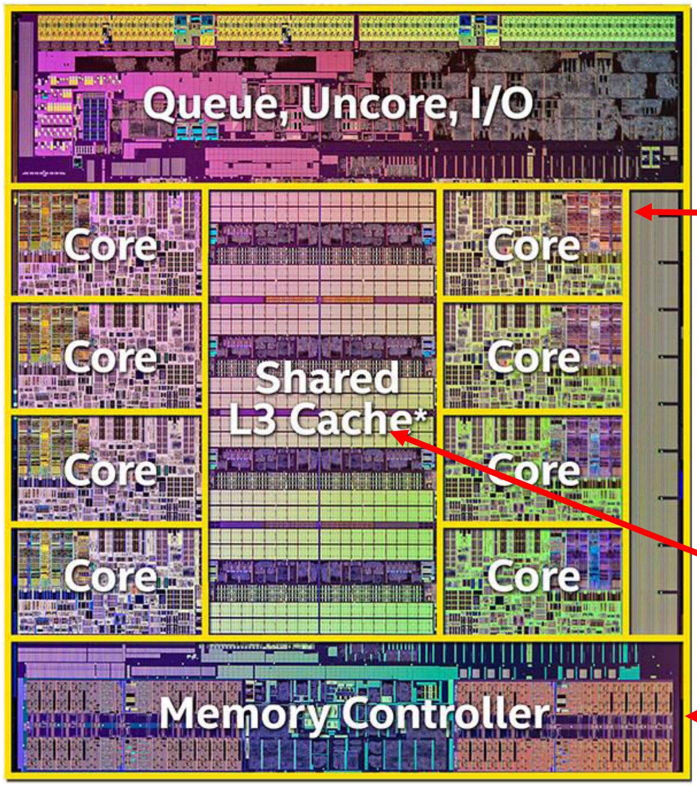
- In the early days of Processor and Dram development, CPUs could run code ***directly*** from Dram, because their speeds were similar.
- However, the illustration below shows that post 1980, CPU speed had raced ahead of Dram speed. Consequently, running programs or accessing data directly from Dram incurred significant delays via the associated **wait states** imposed by the slow Drams, so all that super fast CPU speed was wasted.

▪ Processor vs Memory Performance



Using Caches to Improve Memory Speed

- To combat this, processors were developed to utilize “on-chip” caches to store “local copies” of information maintained in Dram. A cache based CPU could access data and instructions at full CPU speed *without* incurring “wait state” penalties, provided of course the data/instruction had been loaded into cache in the first instance.
- The bigger the cache, the better the “hit” rate and the better the performance.



(2015) 8 Core I7 processor with shared Cache and Memory controller

Each core has a 32kB Instruction and 32kB Data Cache (Level 1 Cache – full speed).

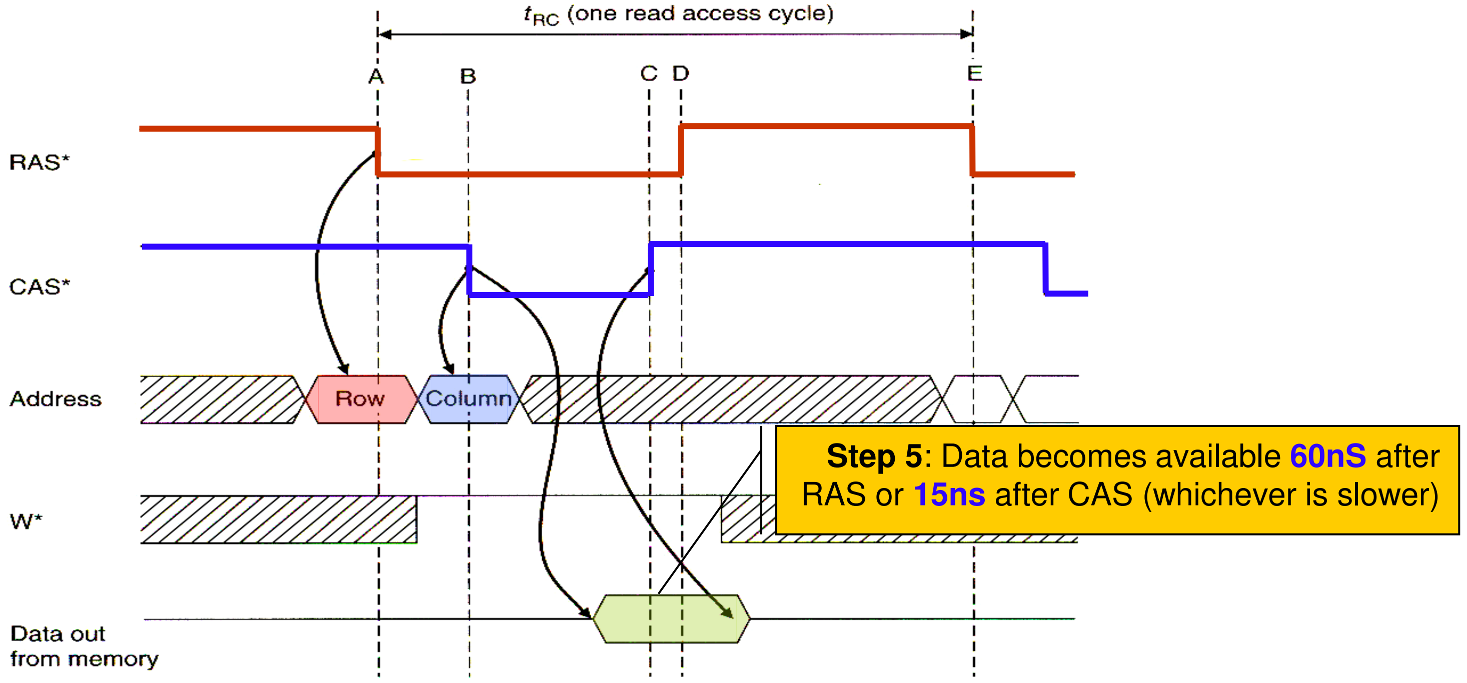
Each core also has 256kB of Level 2 cache which runs slower but is still faster than Dram and caches data and code local the that core.

On Chip 20MB shared Level 3 Cache (giving 2.5MB per core)

Cache/Dram controller

Dram Developments - Fast Page Mode Drams

- The big problem then was how to get data from Dram into the cache as fast as possible, i.e. “fill the cache”. First generation Drams needed both a Ras* and a Cas* signal each time memory was access making them slow.

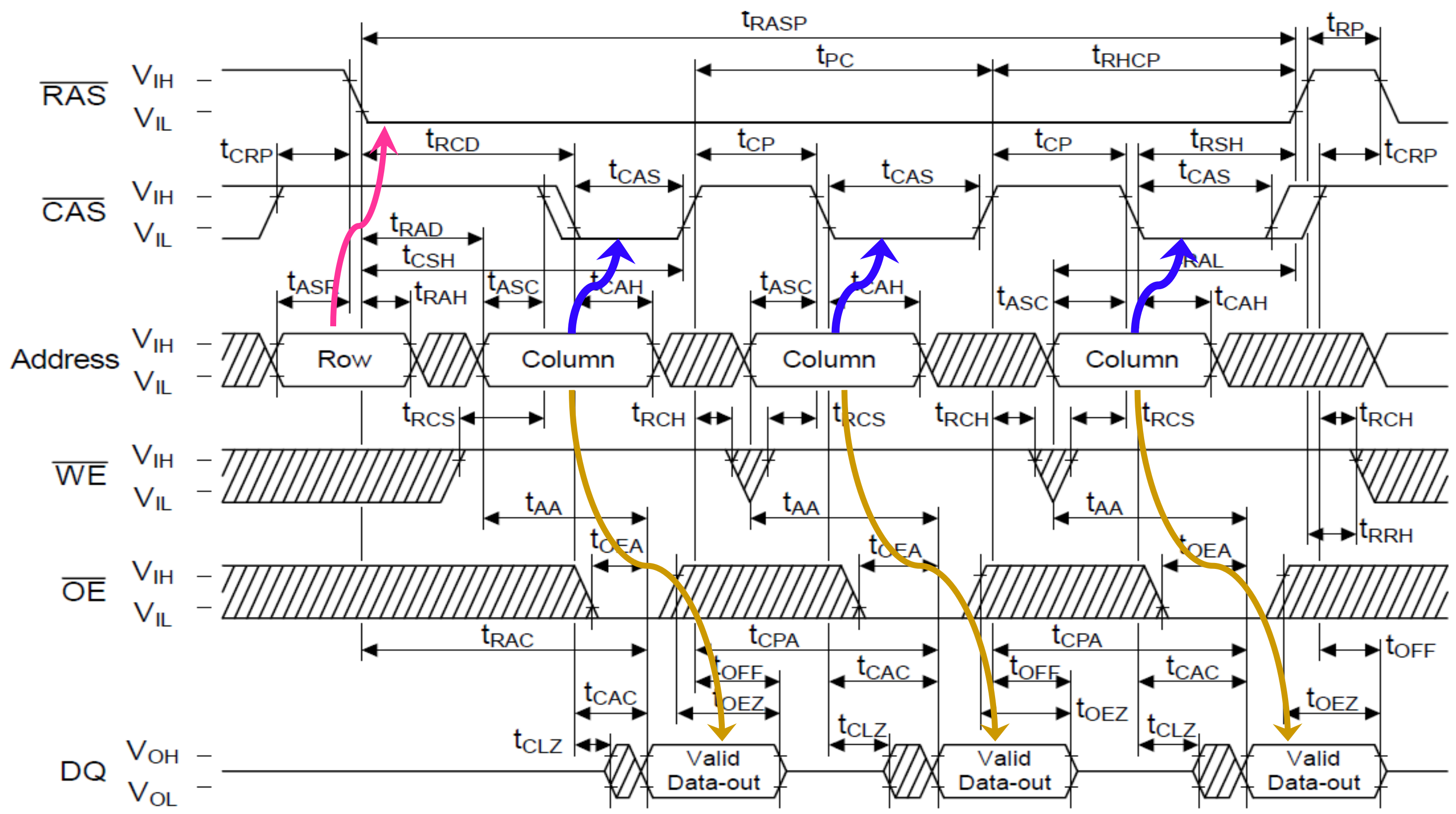


Step 5: Data becomes available **60nS** after RAS or **15ns** after CAS (whichever is slower)

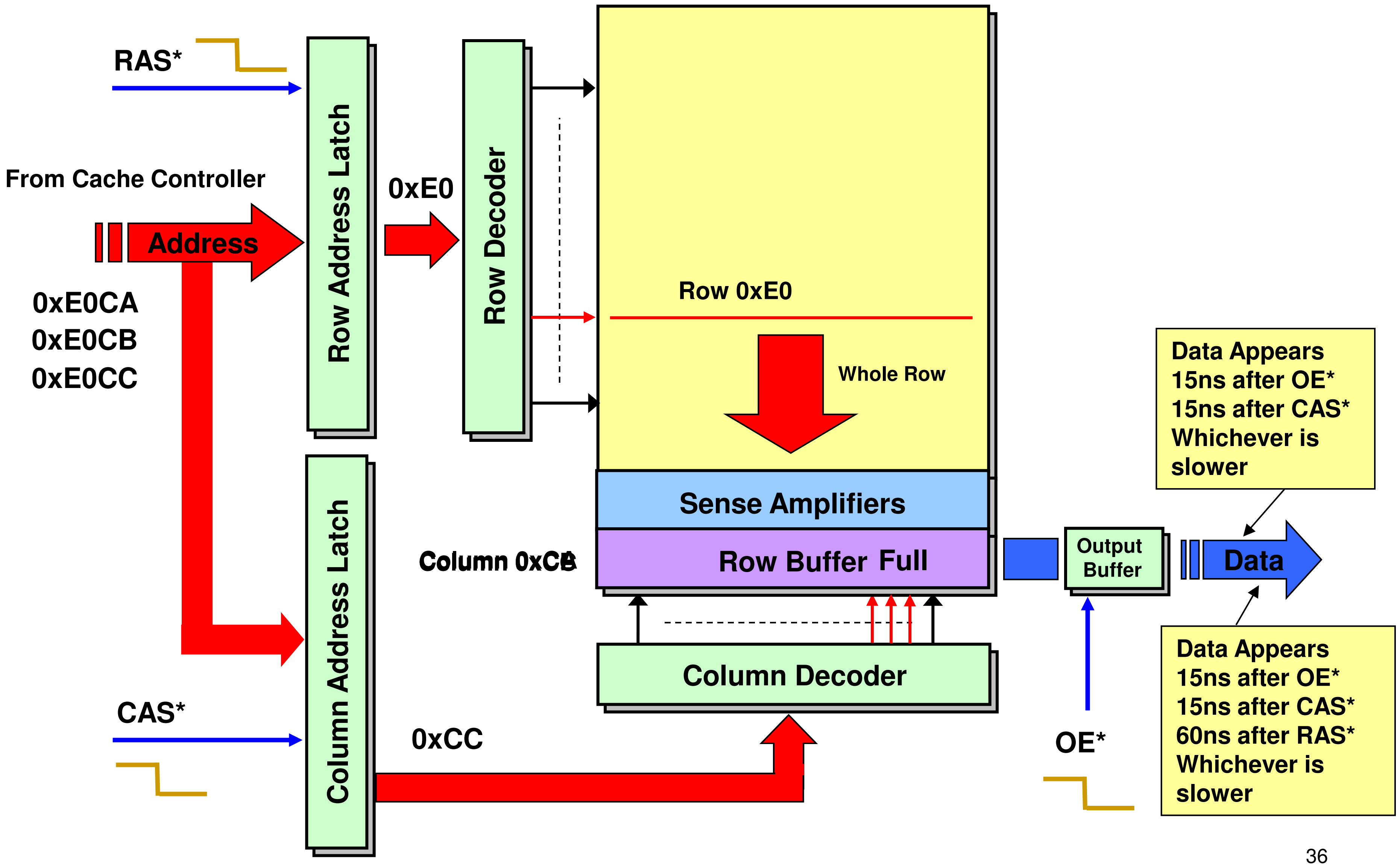
- However by noting that the access time from Ras* (60ns) was significantly longer than that from Cas* (15ns) a reduction in access times could be obtained by using data already in the Dram row buffer.
- This lead to the development of Fast Page Mode Drams in the late 90’s

Fast Page Mode Asynchronous Drams

- In page mode, the row buffer of the DRAM can be kept "**open**" by holding RAS low while performing multiple reads or writes with separate CAS pulses/addresses.
- A simplified timing diagram is given below illustrating the concept.
- Now a cache/memory controller could be designed to take advantage of this and improve the fill speed of a cache by perhaps 300%. New data is available every 40ns using this approach (vs every 110ns using the RAS/CAS approach).



Fast Page Mode Asynchronous 64k Dram Architecture


RAS*
From Cache Controller
Address
0xE0CA
0xE0CB
0xE0CC
Row Address Latch
0xE0
Row Decoder
Row 0xE0
Whole Row
Sense Amplifiers
Row Buffer Full
Column 0xCB
Column Address Latch
CAS*
0xCC
Column Decoder
Output Buffer
Data
OE*
Data Appears 15ns after OE* 15ns after CAS* Whichever is slower
Data Appears 15ns after OE* 15ns after CAS* 60ns after RAS* Whichever is slower