Low Latency and Low Cost DRAM Architecture

Quick memory recap

But why exactly?

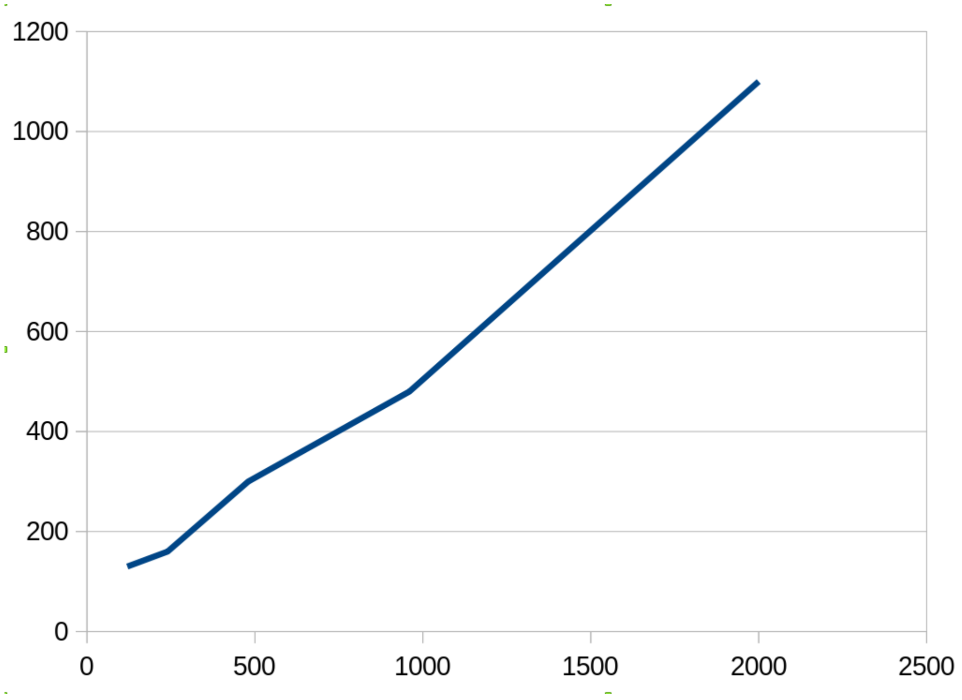
But why exactly?

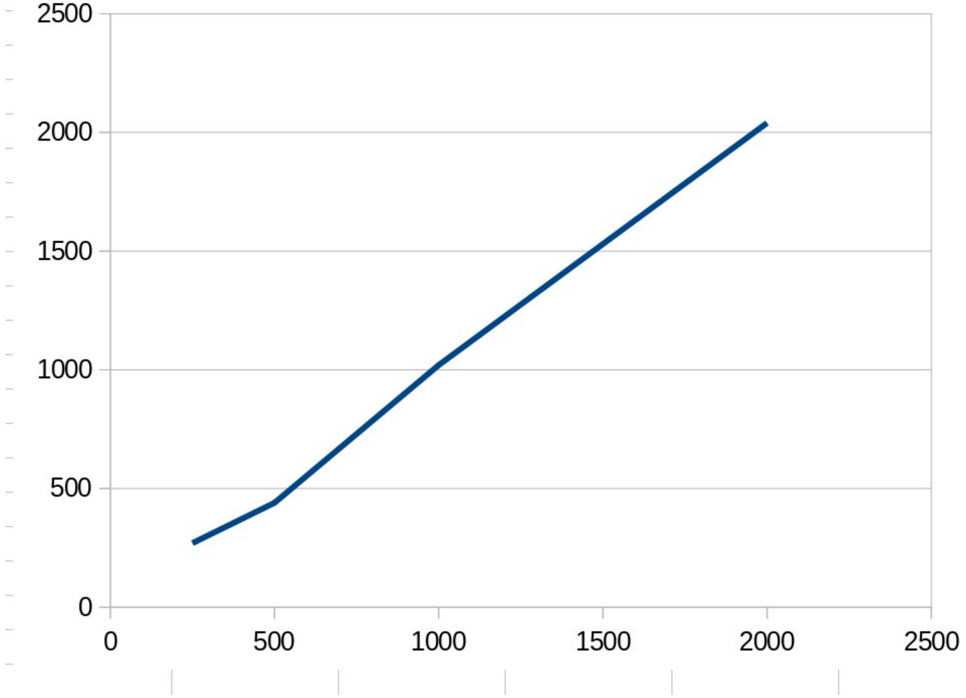
- In SRAM it’s intuitive. We pay more for better speeds.

But why exactly?

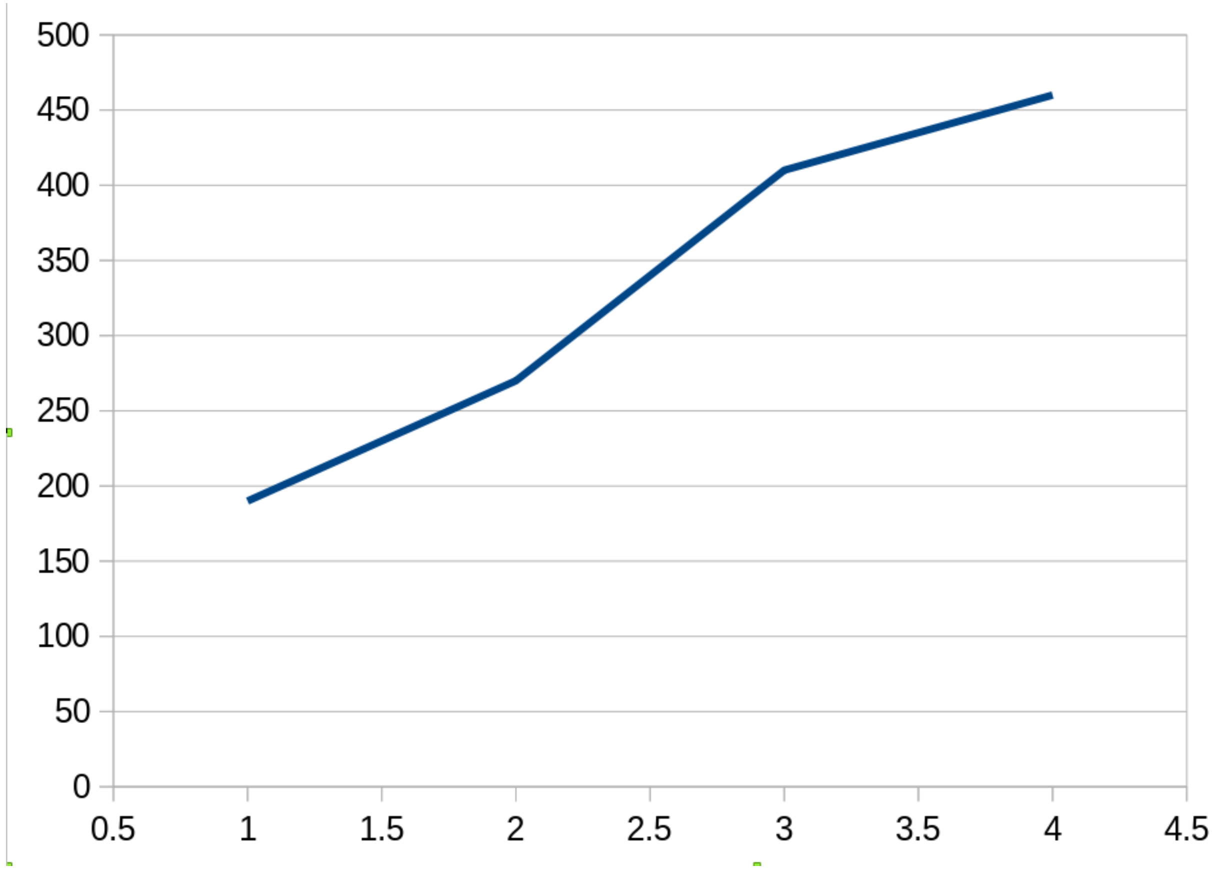
- In SRAM it’s intuitive. We pay more for better speeds.
- But why should it happen in other storage media? Let’s look at couple examples.

Case study. Flash based media

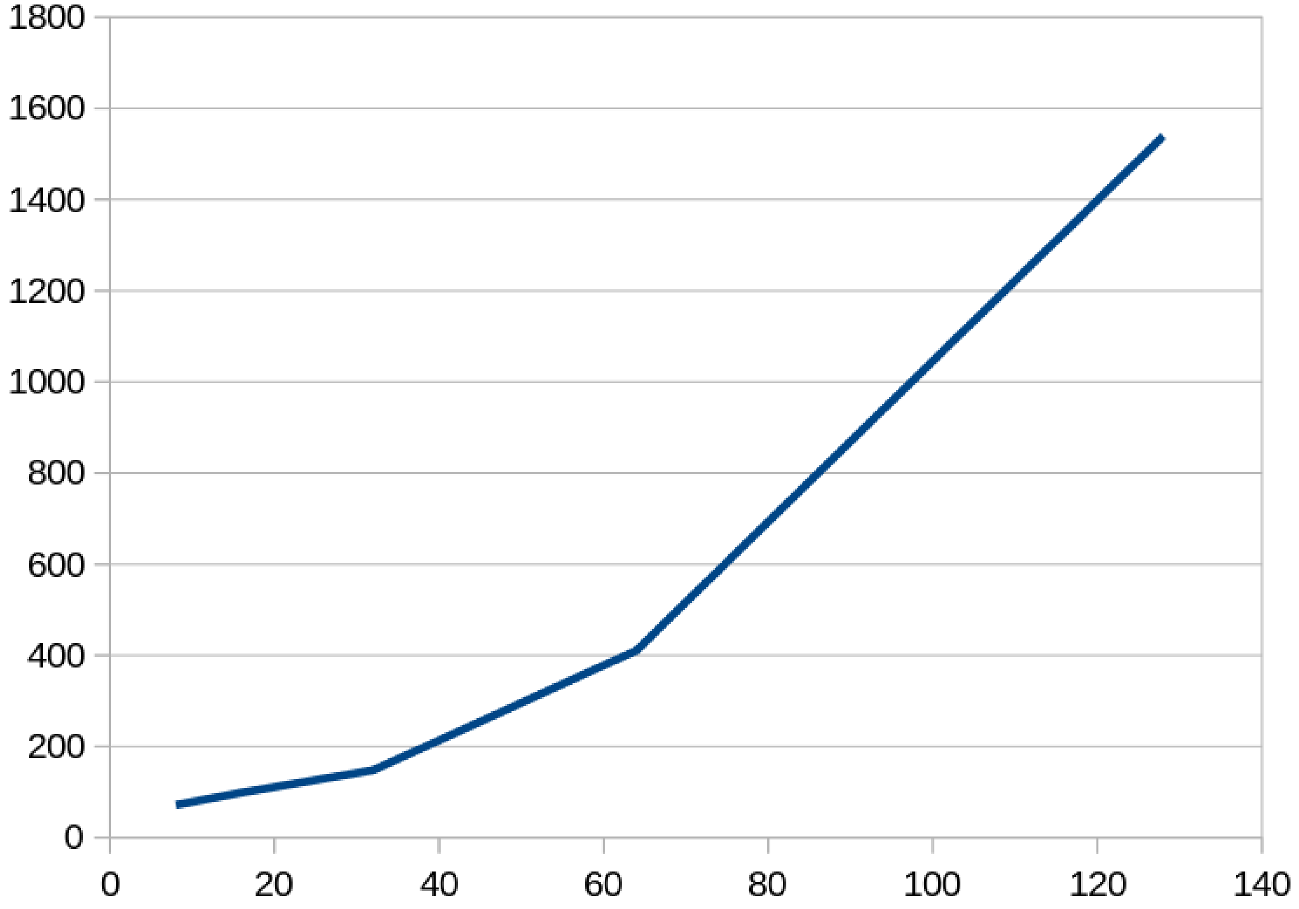




Case study. HDD’s



Case study. DRAM



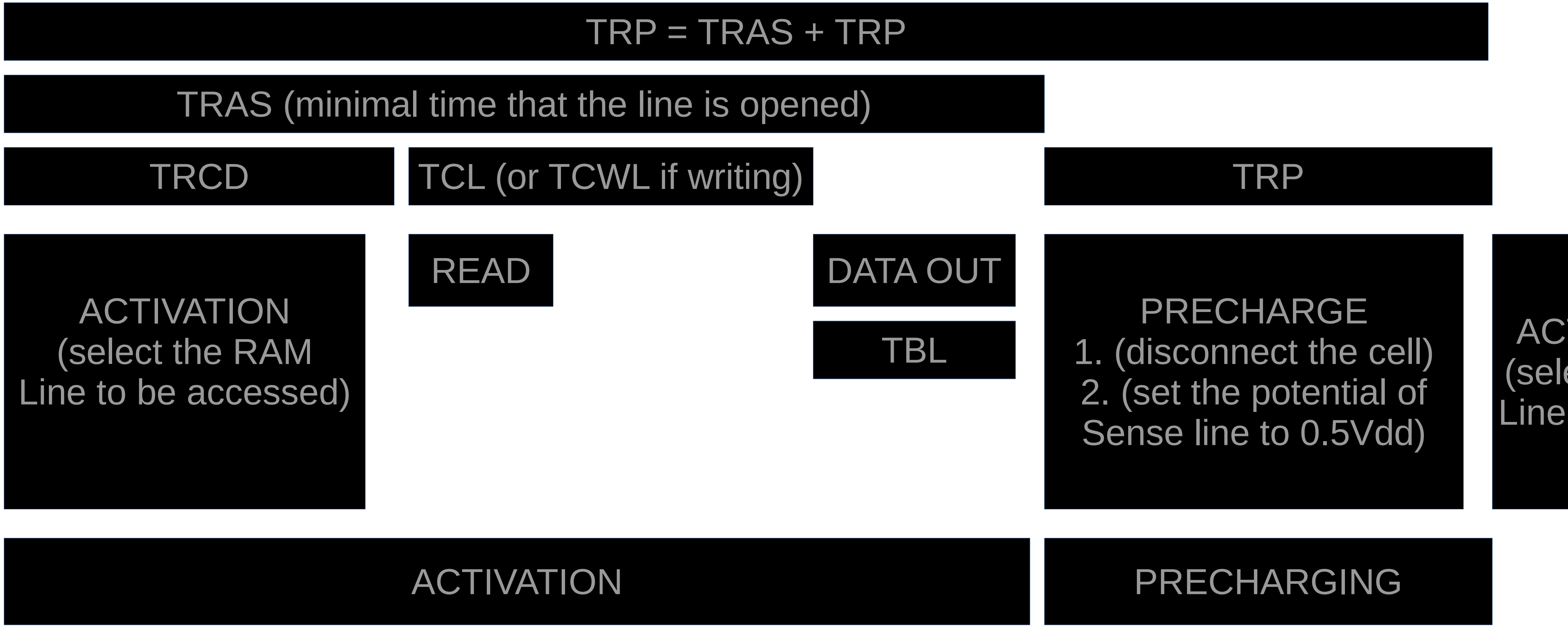
What makes DRAM expensive?

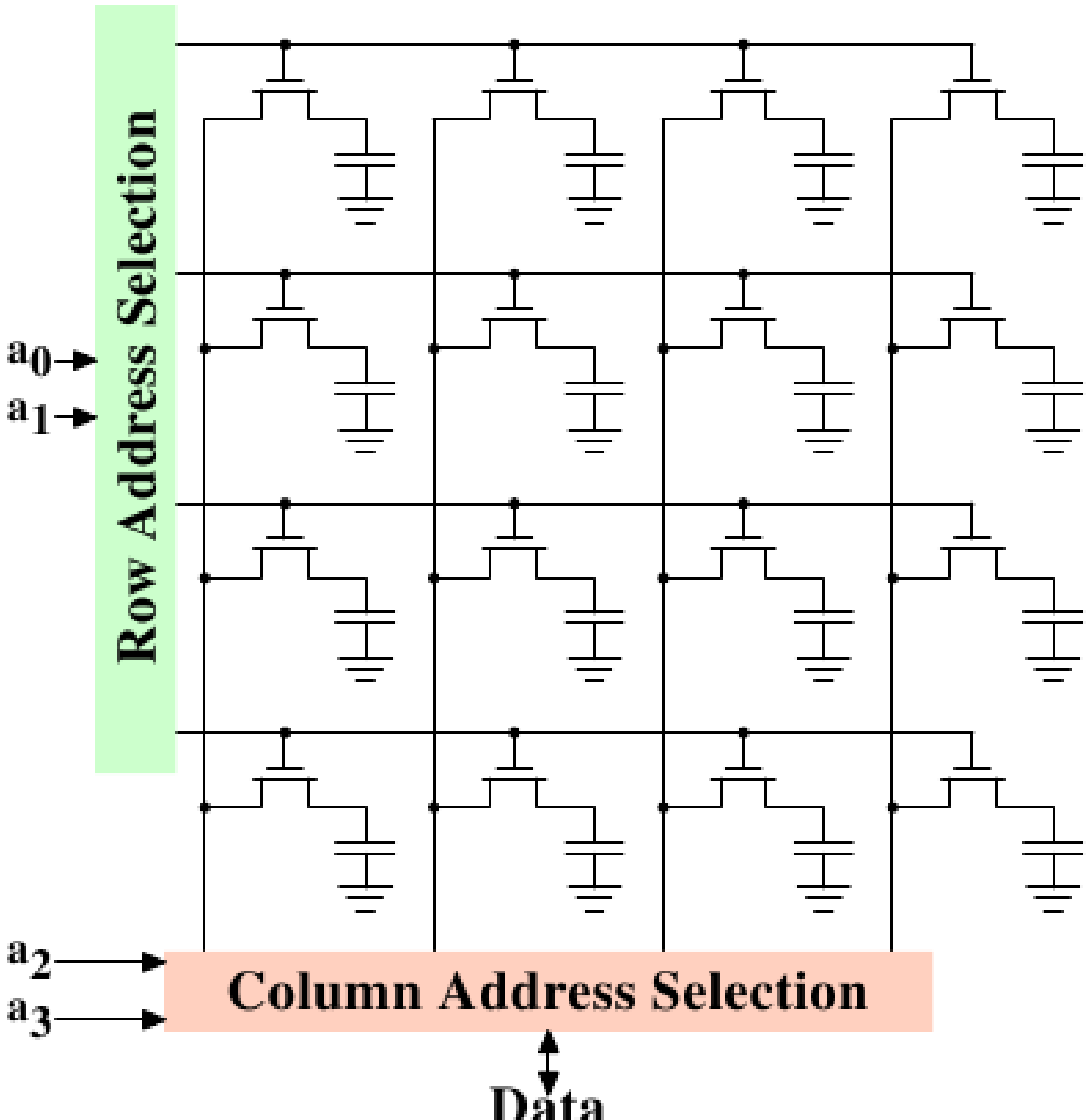
What makes DRAM expensive?

- What makes other stuff cheaper.

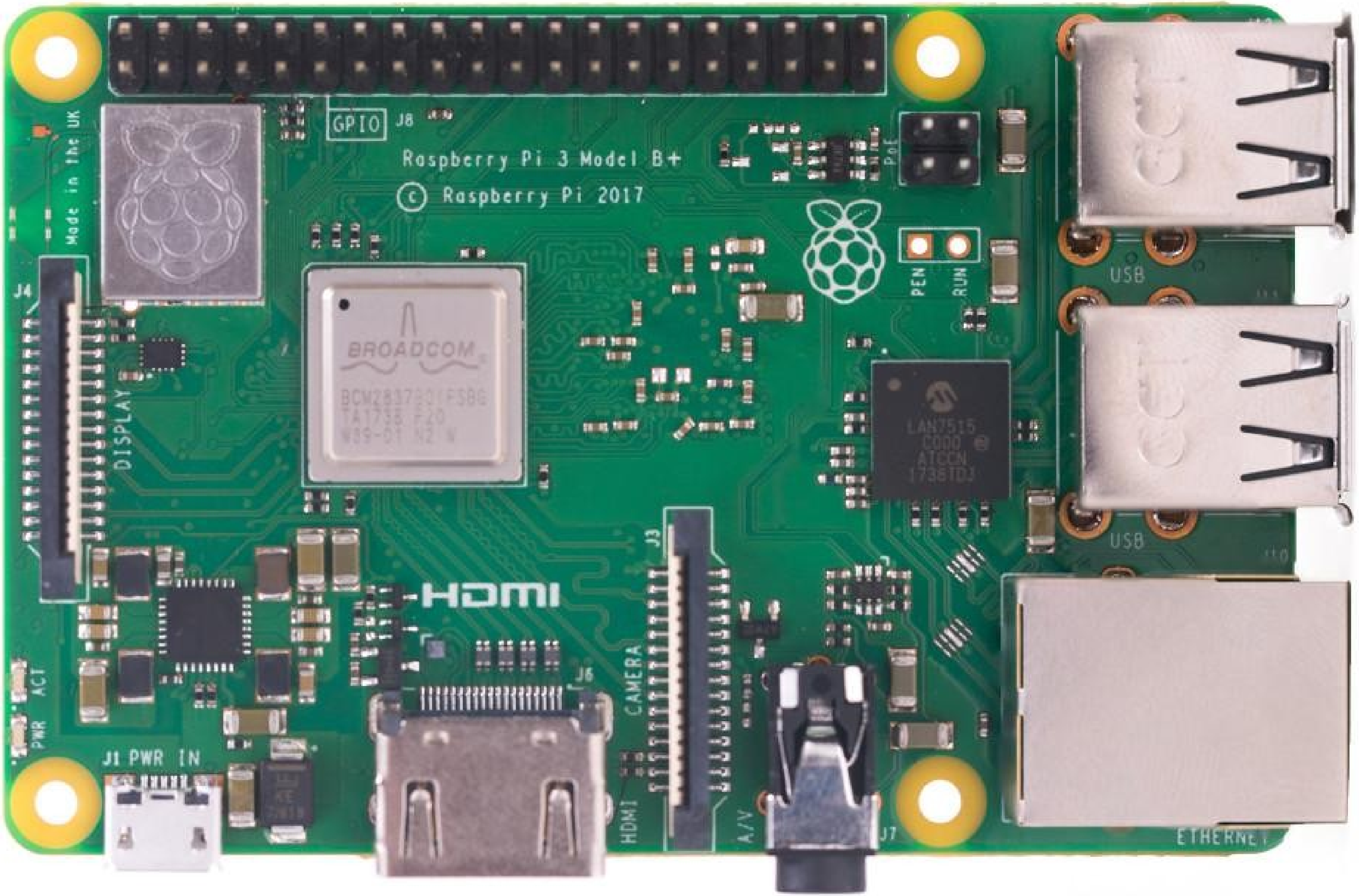
Quick RAM recap

RAM operations

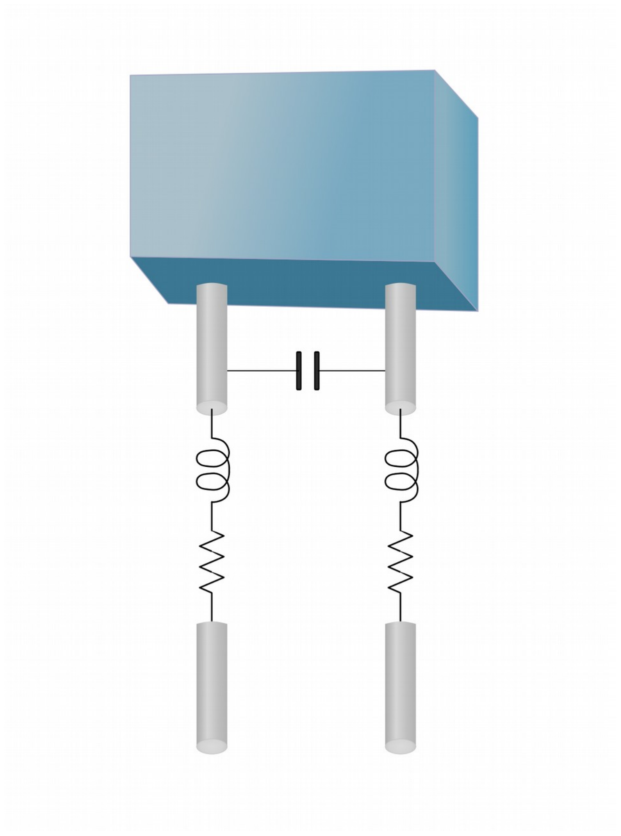

TRP = TRAS + TRP
TRAS (minimal time that the line is opened)
TRCD
TCL (or TCWL if writing)
TRP
ACTIVATION (select the RAM Line to be accessed)
READ
DATA OUT
TBL
PRECHARGE 1. (disconnect the cell) 2. (set the potential of Sense line to 0.5Vdd)
AC (sele Line
ACTIVATION
PRECHARGING


Row Address Selection
a_0
a_1
a_2
a_3
Column Address Selection
Data

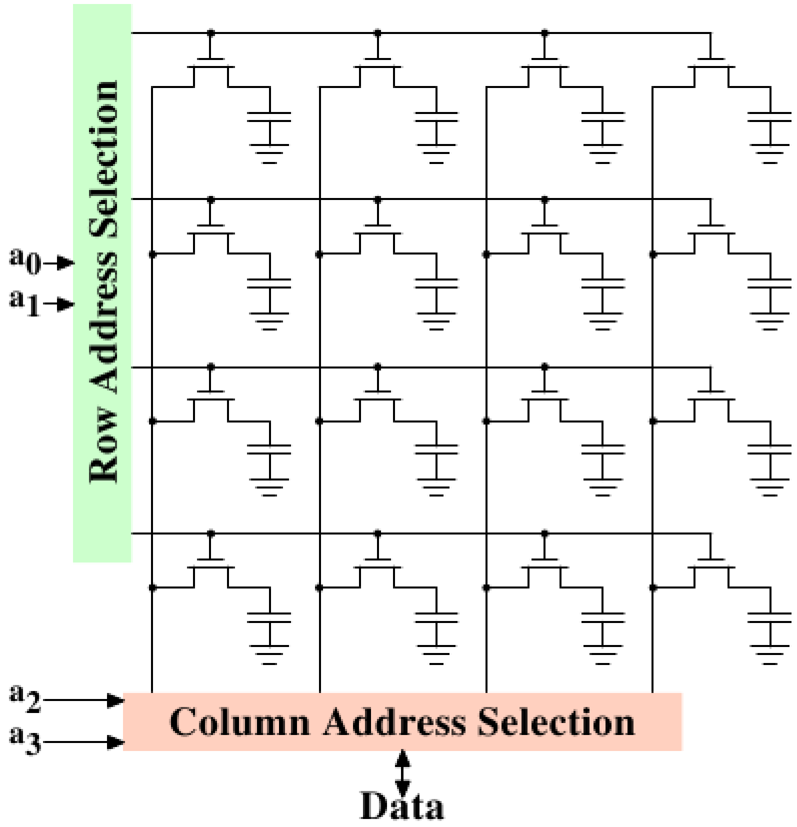
Quick junior high school physics recap


GPIO
J8
Raspberry Pi 3 Model B+
© Raspberry Pi 2017
Made in the UK
PoE
PEN
RUN
USB
USB
BROADCOM
BCM2837B0IFSBG
TA1738 F20
W39-01 N2 W
LAN7515
C000
ATCCN
1738IDJ
J4
DISPLAY
HDMI
J3
CAMERA
J6
ACT
PWR
J1 PWR IN
HDMI
A/V
J7
ETHERNET

Parasitic capacitance is about to hit the fan

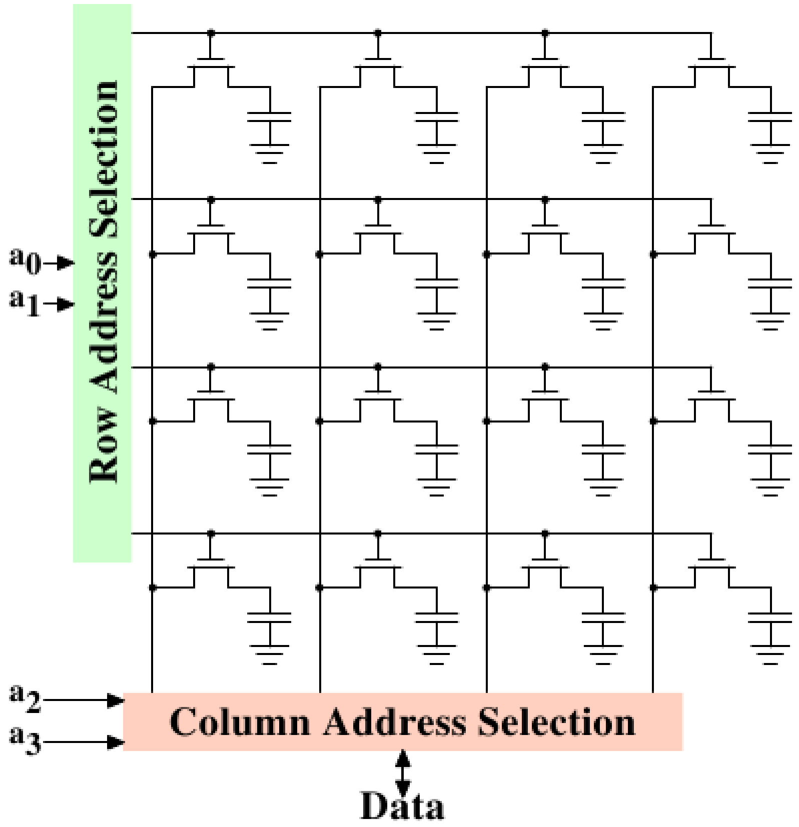


Parasitic capacitance is about to hit the fan



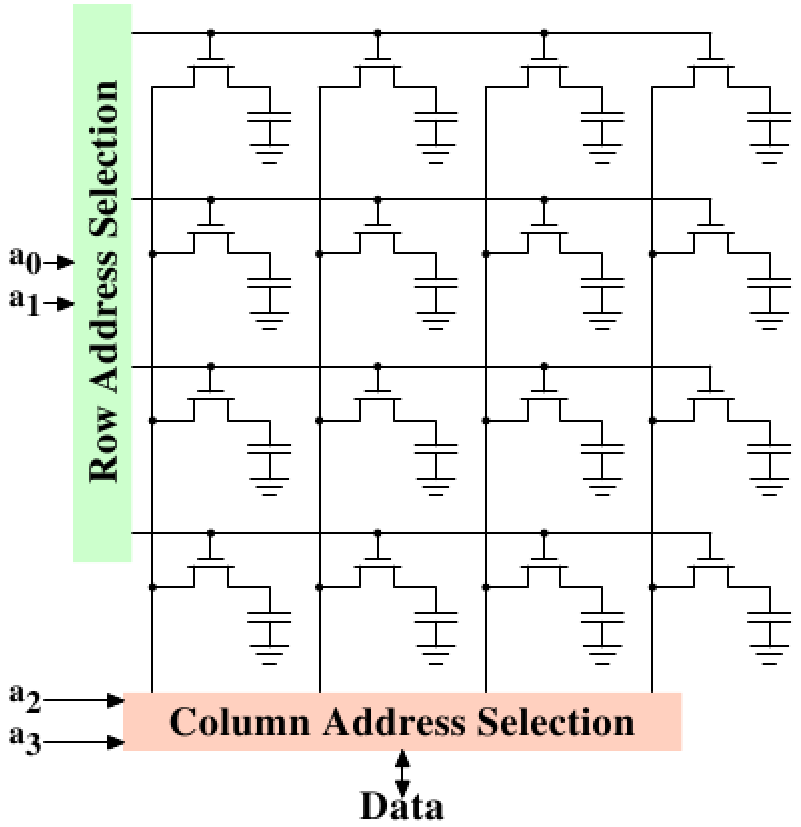
Parasitic capacitance is about to hit the fan

- We can see a lot of places where the capacitance becomes an issue



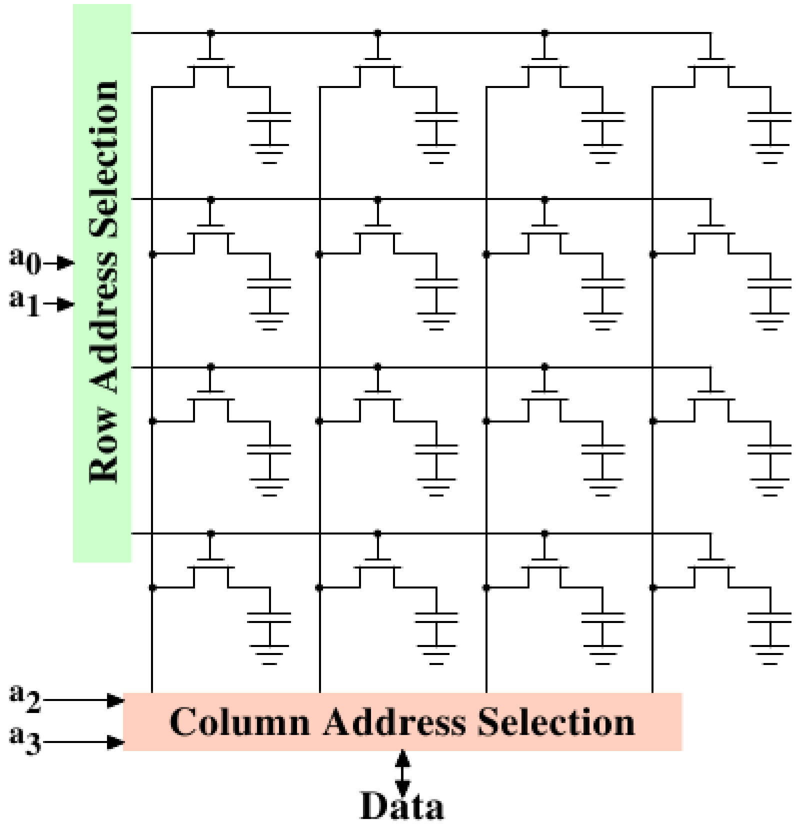
Parasitic capacitance is about to hit the fan

- We can see a lot of places where the capacitance becomes an issue
- Can we fix it?



Parasitic capacitance is about to hit the fan

- We can see a lot of places where the capacitance becomes an issue
- Can we fix it?
- Sure! Let’s see how



Parasitic capacitance is about to hit the fan

(We would like to minimize I)

$$i = C\frac{dv}{dt}$$

High resolution mathematical formulas brought to you by stealing raster graphics from Wikipedia

Parasitic capacitance is about to hit the fan

(We would like to minimize I)

- Idea 1. Minimize dv.

$$i = C\frac{dv}{dt}$$

High resolution mathematical formulas brought to you by stealing raster graphics from Wikipedia

Parasitic capacitance is about to hit the fan

(We would like to minimize I)

- Idea 1. Minimize dv.
- Bad idea. Why?

$$i = C\frac{dv}{dt}$$

High resolution mathematical formulas brought to you by stealing raster graphics from Wikipedia

Parasitic capacitance is about to hit the fan

(We would like to minimize I)

- Idea 2. Increase dt.

$$i = C\frac{dv}{dt}$$

High resolution mathematical formulas brought to you by stealing raster graphics from Wikipedia

Parasitic capacitance is about to hit the fan

(We would like to minimize I)

- Idea 2. Increase dt.
- Sure. Who needed fast RAM

$$i = C\frac{dv}{dt}$$

High resolution mathematical formulas brought to you by stealing raster graphics from Wikipedia

Parasitic capacitance is about to hit the fan

(We would like to minimize I)

- Idea 3. Decrease C.

$$i = C\frac{dv}{dt}$$

High resolution mathematical formulas brought to you by stealing raster graphics from Wikipedia

Parasitic capacitance is about to hit the fan

(We would like to minimize I)

- Idea 3. Decrease C.
- Sure. Let’s look at how C is calculated

$$i = C\frac{dv}{dt}$$

High resolution mathematical formulas brought to you by stealing raster graphics from Wikipedia

Parasitic capacitance is about to hit the fan

(We would like to minimize I)

- Idea 3. Decrease C.
- Sure. Let’s look at how C is calculated

$$i = C\frac{dv}{dt}$$

High resolution mathematical formulas brought to you by stealing raster graphics from Wikipedia

$$\frac{\pi\varepsilon\ell}{\operatorname{arcosh}\left(\frac{d}{2a}\right)} = \frac{\pi\varepsilon\ell}{\ln\left(\frac{d}{2a}+\sqrt{\frac{d^2}{4a^2}-1}\right)}$$

Parasitic capacitance is about to hit the fan

(We would like to minimize I)

- Idea 3. Decrease C.
- Sure. Let's look at how C is calculated
  - Next best approximation uses elliptic integrals

$$i = C\frac{dv}{dt}$$

High resolution mathematical formulas brought to you by stealing raster graphics from Wikipedia

Parasitic capacitance is about to hit the fan

(We would like to minimize I)

- Idea 3. Decrease C.
- Sure. Let’s look at how C is calculated
  - Next best approximation uses elliptic integrals
  - We use the power of “Finalista Ślązaczka z fizyki” to pull even simpler approximations out of our ~~ass~~ sleeve

$$i = C\frac{dv}{dt}$$

High resolution mathematical formulas brought to you by stealing raster graphics from Wikipedia

Parasitic capacitance is about to hit the fan

(We would like to minimize I)

- Idea 3. Decrease C.
- Sure. Let’s look at how C is calculated

C = pA/d

$$i = C\frac{dv}{dt}$$

High resolution mathematical formulas brought to you by stealing raster graphics from Wikipedia

Parasitic capacitance is about to hit the fan

(We would like to minimize I)

- Idea 3. Decrease C.
- Sure. Let’s look at how C is calculated

  C = pA/d
- That’s more like it. Now, what do these mean?

$$i = C\frac{dv}{dt}$$

High resolution mathematical formulas brought to you by stealing raster graphics from Wikipedia

Parasitic capacitance is about to hit the fan

- (We would like to minimize C) $C = pA/d$
- Idea 1. Decrease p.

Parasitic capacitance is about to hit the fan

- (We would like to minimize C) $C = pA/d$
- Idea 1. Decrease p. Unlikely. That would require modifications to the production process

Parasitic capacitance is about to hit the fan

- (We would like to minimize C) $C = pA/d$
- Idea 1. Decrease p. Unlikely. That would require modifications to the production process
- Idea 2. Decrease A.

Parasitic capacitance is about to hit the fan

- (We would like to minimize C) C = pA/d
- Idea 1. Decrease p. Unlikely. That would require modifications to the production process
- Idea 2. Decrease A. Shorter/thinner wires. Both unlikely for obvious reasons.

Parasitic capacitance is about to hit the fan

- (We would like to minimize C) C = pA/d
- Idea 1. Decrease p. Unlikely. That would require modifications to the production process
- Idea 2. Decrease A. Shorter/thinner wires. Both unlikely for obvious reasons.
- Idea 3. Increase d.

Parasitic capacitance is about to hit the fan

- (We would like to minimize C) $C = pA/d$
- Idea 1. Decrease p. Unlikely. That would require modifications to the production process
- Idea 2. Decrease A. Shorter/thinner wires. Both unlikely for obvious reasons.
- Idea 3. Increase d. Sure, who needed small and efficient DRAM dies.

This concludes the presentation

Low Latency and Low Cost DRAM Architecture is impossible.

(Standing ovation)

Po prostu powiększ swoje pamięci DRAM

Wybierz rozmiar:

○ 1GiB

○ 2GiB

○ 4GiB

[✓ 16GiB] (Ostrożnie!)

POTWIERDŹ

But wait, there is more!

- What if decreasing A in $C=pA/d$ was a valid solution?

But wait, there is more!

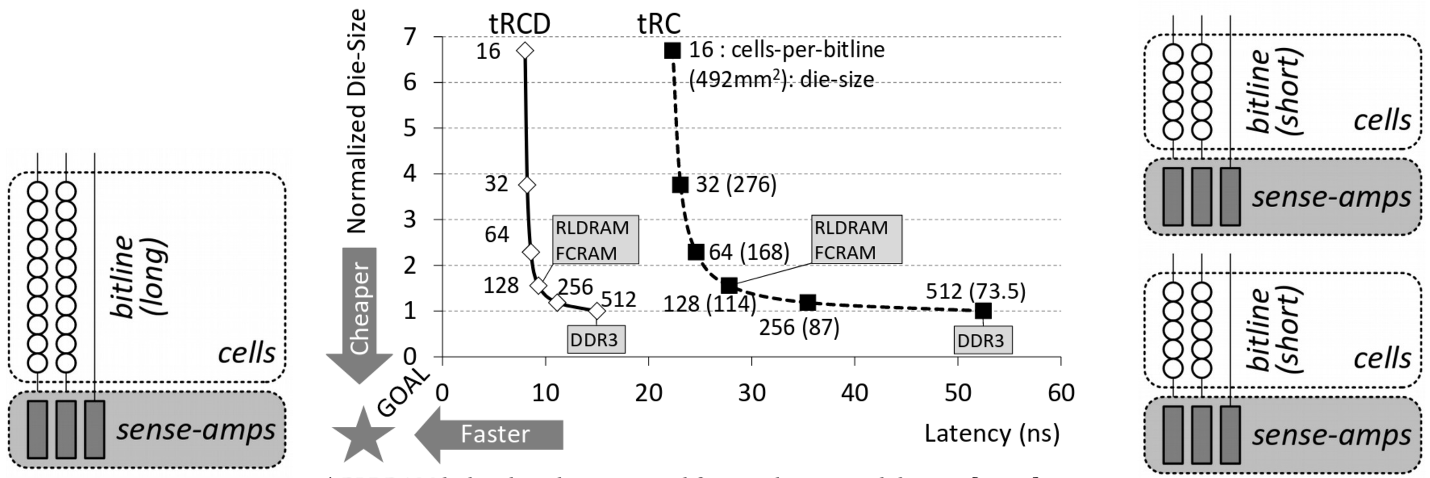
- What if decreasing A in $C=pA/d$ was a valid solution?
- Sure, we get less memory per an amplifier circuit.

But wait, there is more!

- What if decreasing A in $C=pA/d$ was a valid solution?
- Sure, we get less memory per an amplifier circuit.
- But we can always add more amplifiers (increase row length)

But wait, there is more!

- What if decreasing A in C=pA/d was a valid solution?
- Sure, we get less memory per an amplifier circuit.
- But we can always add more amplifiers (increase row length)



(b) Cost Opt.

† RLDRAM bitline length is estimated from its latency and die-size [21, 27].
† The reference DRAM is 55nm 2GB DDR3 [39].

(a) Latency Opt.

Once again...

- Memories are cheap, fast, capacious (choose two out of three).

Once again...

- Memories are cheap, fast, capacious (choose two out of three).
- What if we could change what two of them we are using?

Once again...

- Memories are cheap, fast, capacious (choose two out of three).
- What if we could change what two of them we are using?

(Obviously we can’t change the cost once we have bought it, so the more awake people in the room have already guessed that sometimes we will choose speed over capacitance)

Sorry, WHAT?

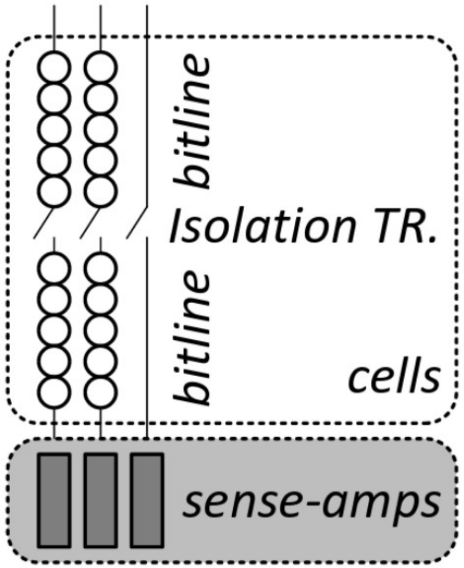
- Introducing the stupidly simple, and super obvious in retrospect trick for faster memory operations.

Sorry, WHAT?

- Introducing the stupidly simple, and super obvious in retrospect trick for faster memory operations – RGB RAM.

Sorry, WHAT?

- Introducing the stupidly simple, and super obvious in retrospect trick for faster memory operations.



(c) Our Proposal

Let’s look at the benefits

Implementation: in-die transfer

Implementation: row decoder

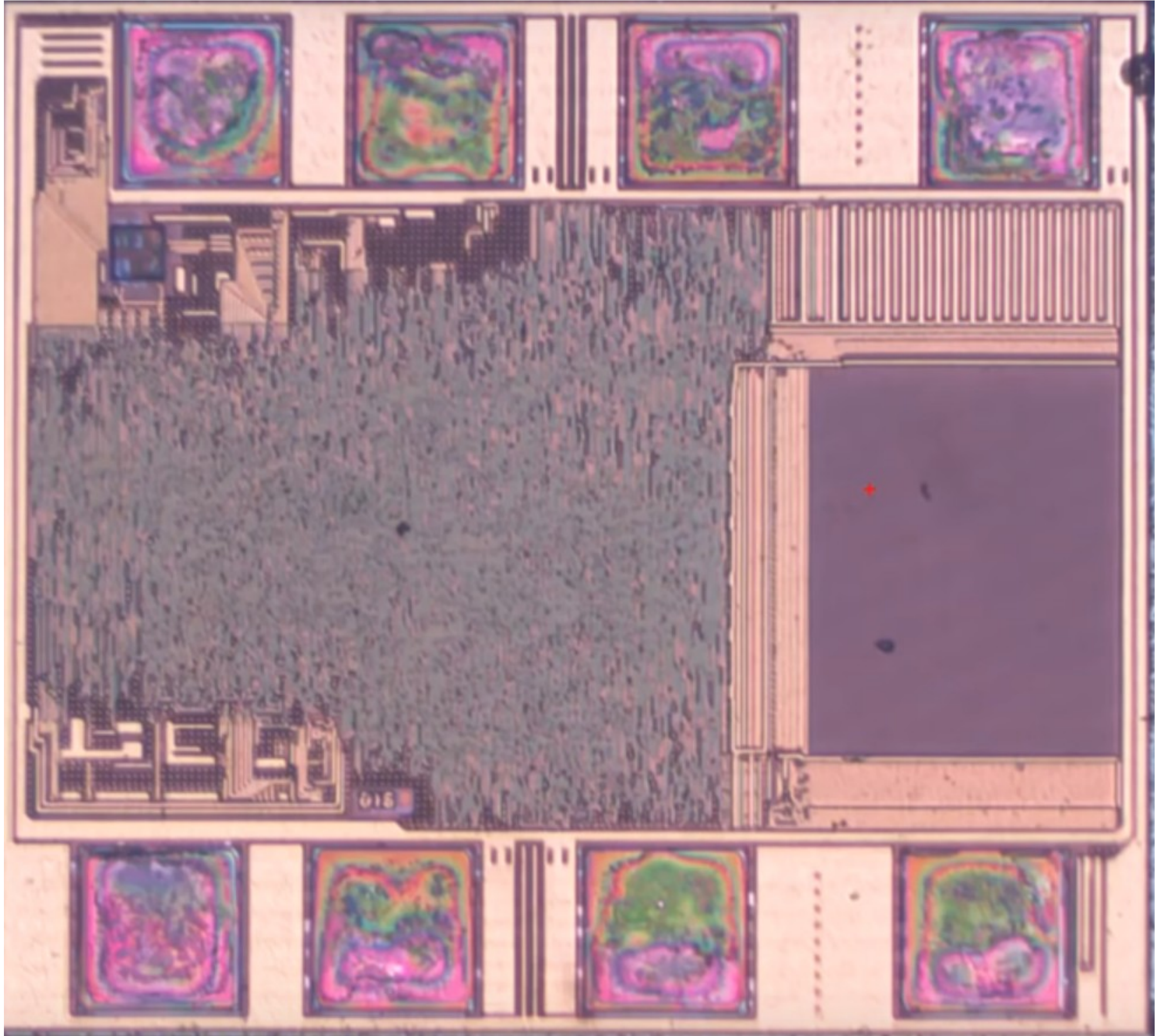
- Predecoding.
- Input address is split to M blocks.
- For each block (size of N) output 2^N wires (a simple decoder)
- At each row an AND gate with M inputs.
- We pay with additional wires for a decreasing the logic at each row.

Implementation: double row decoder

- For on-die data transfers we need to be able to select two rows at once.
- A naive approach dictates that we double the whole row decoder
- Why is this approach bad?

Implementation: double row decoder

- For on-die data transfers we need to be able to select two rows at once.
- Since we only transfer between far and near segment, we only double the lines for the shorter one
- Only 0.33% size penalty


018

Implementation: metadata

This would mark the end of the hardware part of the presentation

Since now we basically have a cache in DRAM, all the juicy topics of cache management map onto managing cache inside of DRAM

Near cache is transparent to the OS

- Approach 1. Simple Caching (SC)

Near cache is transparent to the OS

- Approach 1. Simple Caching (SC)
- We apply LRU to the DRAM accesses. We categorize all accesses as one of three
  - Sense amp hit
  - Near segment hit
  - Near segment miss

Near cache is transparent to the OS

- Approach 1. Simple Caching (SC)
- We apply LRU to the DRAM accesses. We categorize all accesses as one of three
  - Sense amp hit – Serve the memory, don’t change LRU
  - Near segment hit
  - Near segment miss

Near cache is transparent to the OS

- Approach 1. Simple Caching (SC)
- We apply LRU to the DRAM accesses. We categorize all accesses as one of three
  - Sense amp hit – Serve the memory, don’t change LRU
  - Near segment hit – fetch from near segment, set as MRU
  - Near segment miss
    - .

Near cache is transparent to the OS

- Approach 1. Simple Caching (SC)
- We apply LRU to the DRAM accesses. We categorize all accesses as one of three
  - Sense amp hit – Serve the memory, don’t change LRU
  - Near segment hit – fetch from near segment, set as MRU
  - Near segment miss – fetch from far segment, set as MRU.
    - Possibly evict LRU from near segment.
    - Possibly dump LRU to far segment if the DRAM line is dirty.

Near cache is transparent to the OS

- Approach 2. Wait minimized caching (WMC)
- Since I should be hospitalized due to my overdose of caffeine, and the fact that overdosing on substances that promise to give you energy actually cuts ones mental ability, I will now read the WMC description from the paper, since I cannot make any sense of it.

Near cache is transparent to the OS

- Approach 3. Benefit Based Caching (BBC)

Near cache is transparent to the OS

- Approach 3. Benefit Based Caching (BBC)
- Same as Simple Caching, but instead of LRU we calculate the benefit of a block being in near segment)

Near cache is transparent to the OS

- Approach 3. Benefit Based Caching (BBC)
- Same as Simple Caching, but instead of LRU we calculate the benefit of a block being in near segment)
- When a far segment is hit, we evict a block with the smallest benefit. On every operation we halve the benefit of all blocks.

Near cache is transparent to the OS

- Approach 3. Benefit Based Caching (BBC)
- Same as Simple Caching, but instead of LRU we calculate the benefit of a block being in near segment)
- When a far segment is hit, we evict a block with the smallest benefit. Every operation we halve the benefit of all blocks.
- Benefit is the amount of cycles saved by the block being in the near segment.

Exposing the cache to OS

- Simply allow OS to access the near region as regular RAM.
- Hope to get increased performance due to lower timings for some memory
- Has the benefit of being a simple replacement, without any change to host system
- Sadly, low performance increase

Exclusive cache

- Use the memory controller to handle decisions on what to put in cache
- Since cache is exclusive, we keep one row clear in order to perform swap operations

Profile based page mapping

- OS controls virtual to physical mapping (this was discussed last week, with the pmap function)
- TL-DRAM informs OS about the areas of physical memory that are in the close region
- Information about the frequency of usage can be obtained during compilation time, or dynamically via hardware counters.

This concludes the software part of the presentation

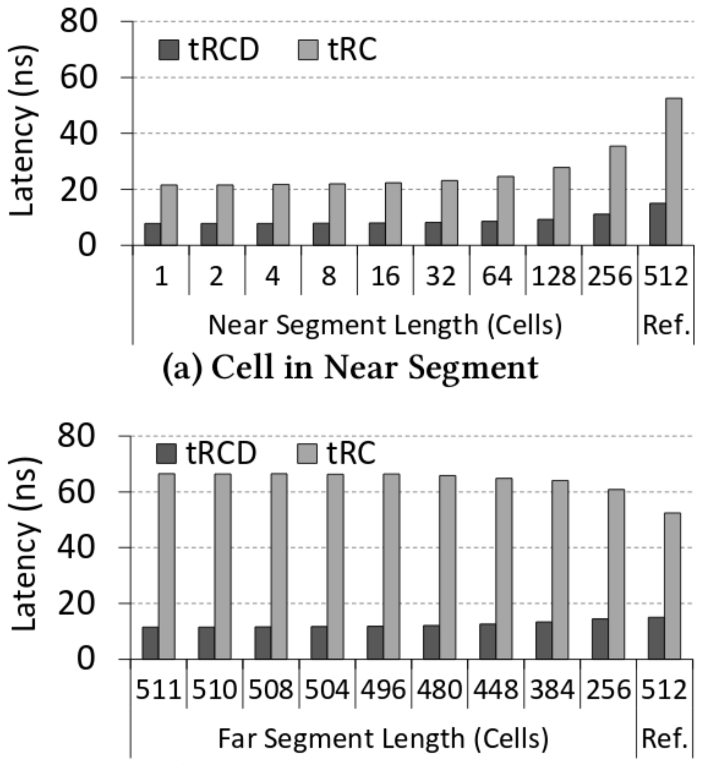
EVALUATION

Hardware setup

- We first need to consider the ratio between the near and far region

Hardware setup

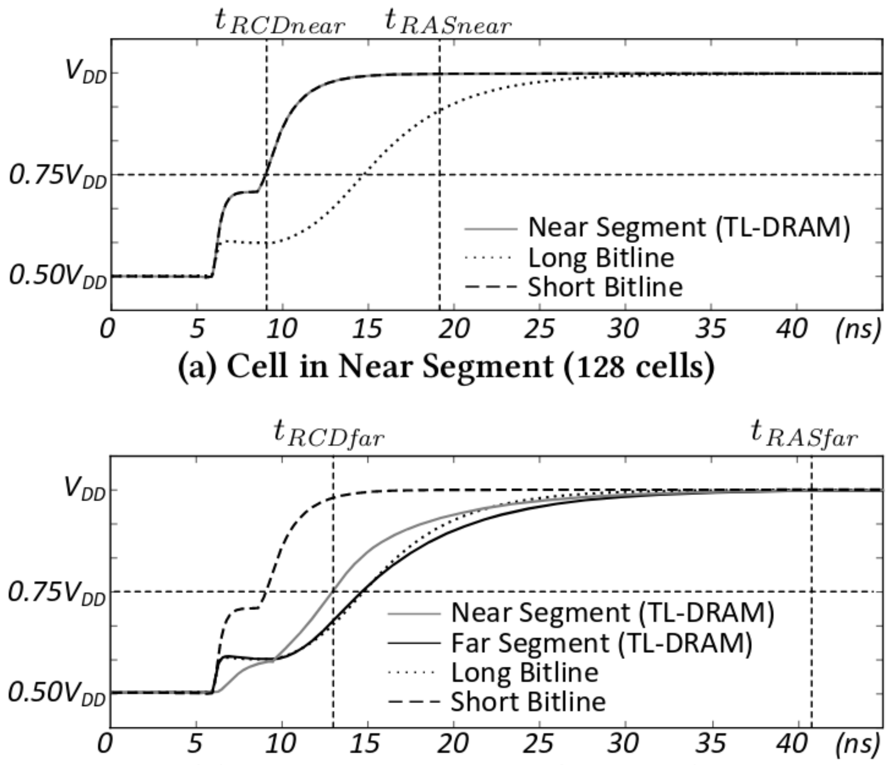
- We first need to set the ratio between the near and far region



(a) Cell in Near Segment

(b) Cell in Far Segment

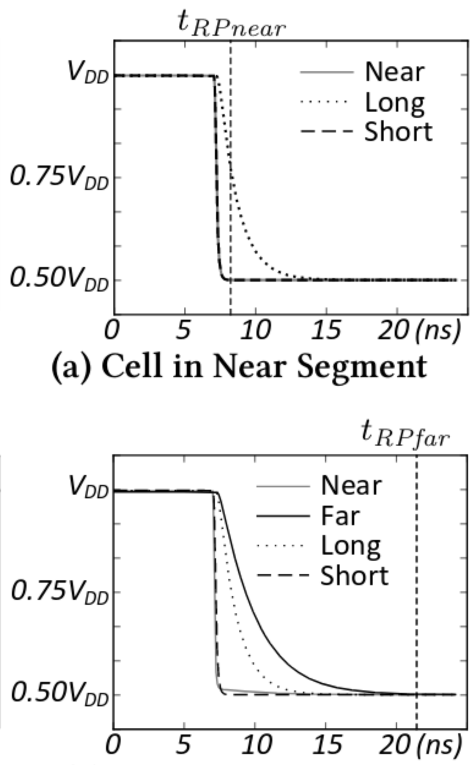
Figure 10. Latency Analysis



(a) Cell in Near Segment (128 cells)

(b) Cell in Far Segment (384 cells)

Figure 11. Activation: Bitline Voltage



(a) Cell in Near Segment

(b) Cell in Far Segment

Figure 12. Precharging

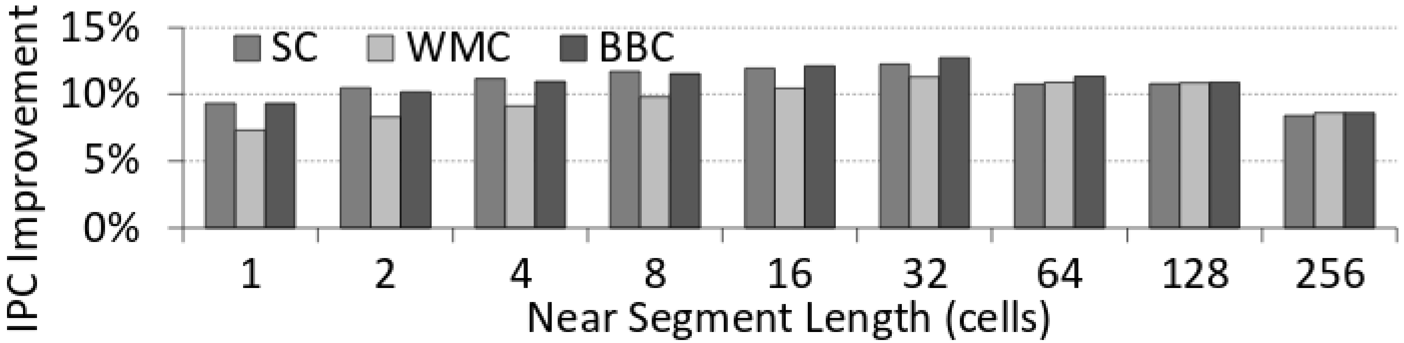
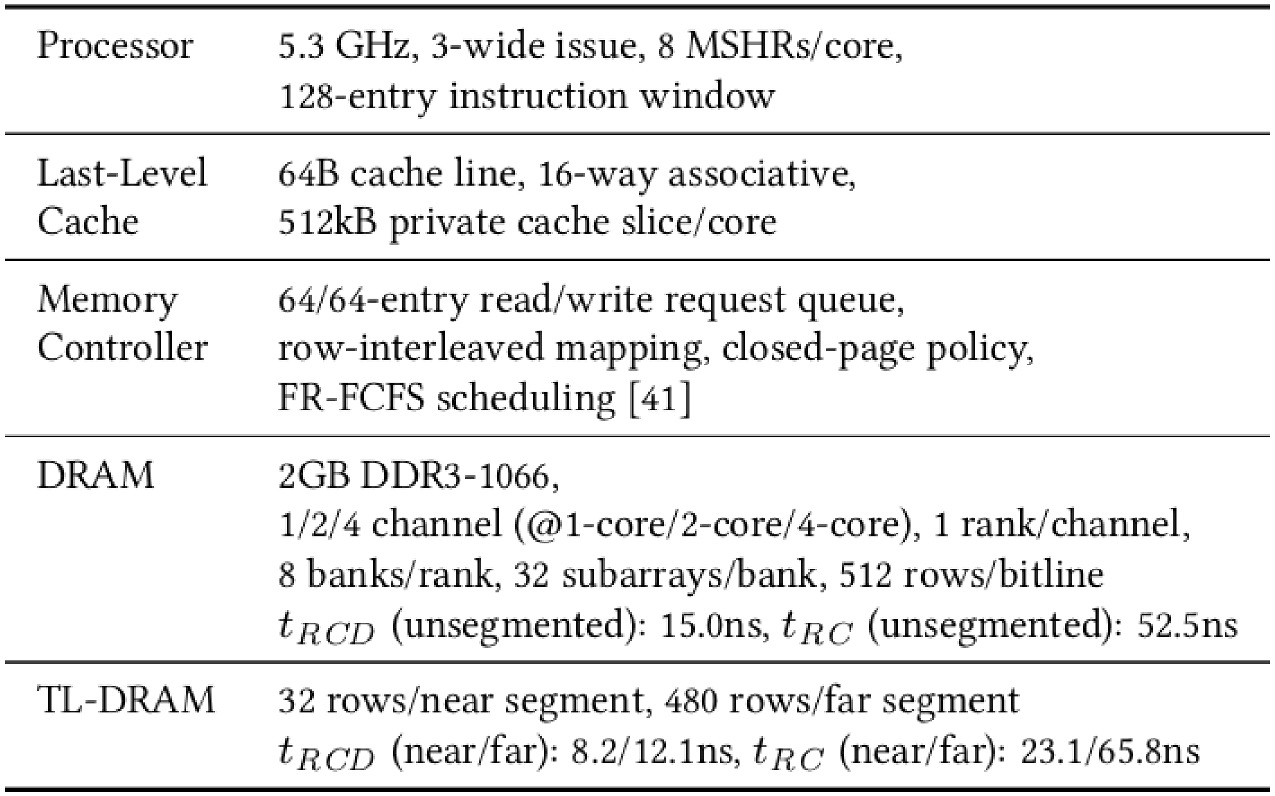


Figure 16. Varying Near Segment Capacity (Inclusive Cache)

Test system specs (SPICE simulation)

Table 4. Evaluated System Configuration



Processor	5.3 GHz, 3-wide issue, 8 MSHRs/core, 128-entry instruction window
Last-Level Cache	64B cache line, 16-way associative, 512kB private cache slice/core
Memory Controller	64/64-entry read/write request queue, row-interleaved mapping, closed-page policy, FR-FCFS scheduling [41]
DRAM	2GB DDR3-1066, 1/2/4 channel (@1-core/2-core/4-core), 1 rank/channel, 8 banks/rank, 32 subarrays/bank, 512 rows/bitline t_{RCD} (unsegmented): 15.0ns, t_{RC} (unsegmented): 52.5ns
TL-DRAM	32 rows/near segment, 480 rows/far segment t_{RCD} (near/far): 8.2/12.1ns, t_{RC} (near/far): 23.1/65.8ns

Table 1. Latency & Die-Size Comparison of DRAMs (Sec 3)

		Short Bitline (Fig 2a)	Long Bitline (Fig 2b)	Segmented Bitline (Fig 2c)	
		Unsegmented	Unsegmented	Near	Far
Length (Cells)		32	512	32	480
Latency	t_{RCD}	Lowest (8.2ns)	High (15ns)	Lowest (8.2ns)	**Low** (12.1ns)
	t_{RC}	Lowest (23.1ns)	High (52.5ns)	Lowest (23.1ns)	Higher (65.8ns)
Normalized Die-Size (Cost)		Highest (3.76)	Lowest (1.00)	Low (1.03)	

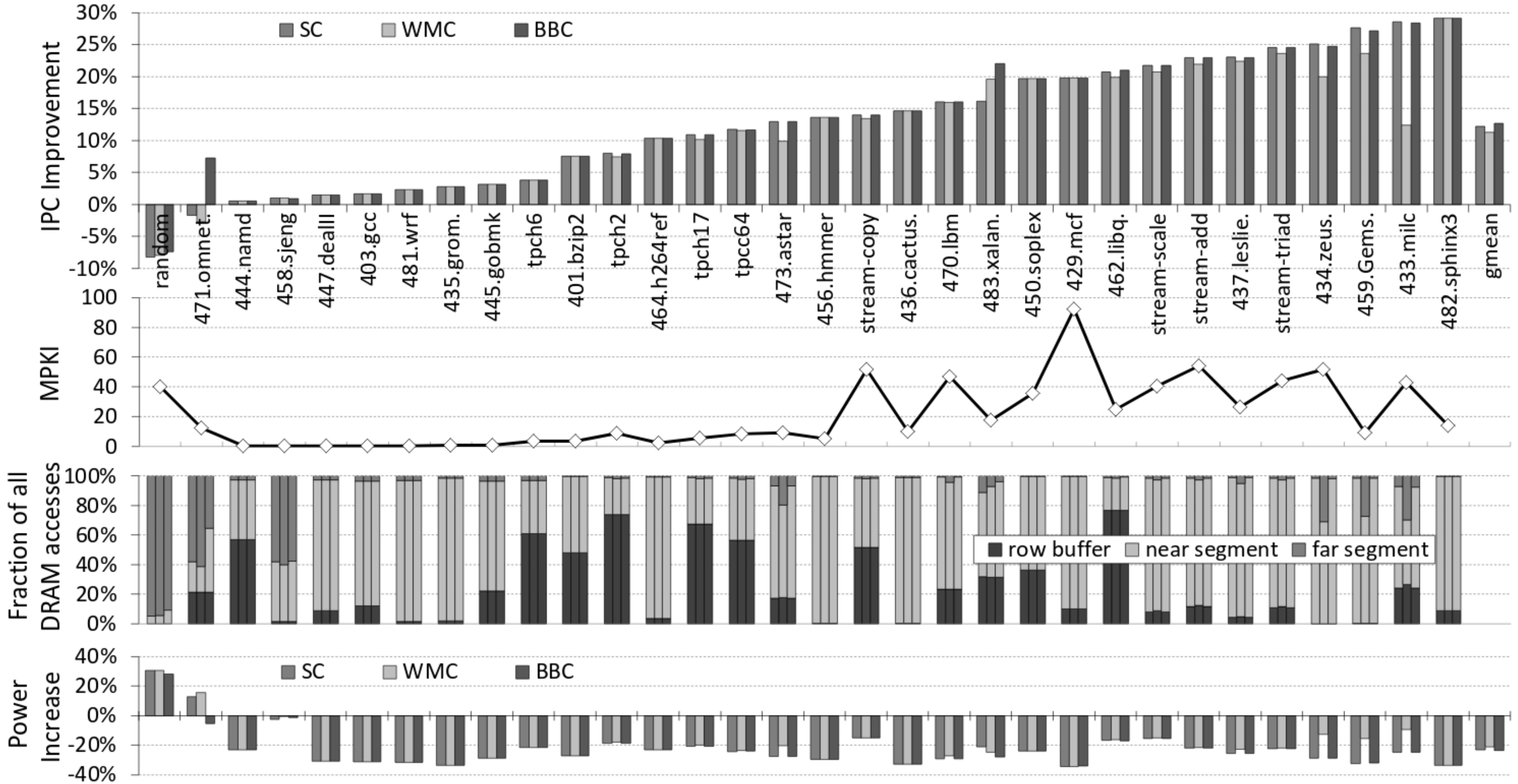
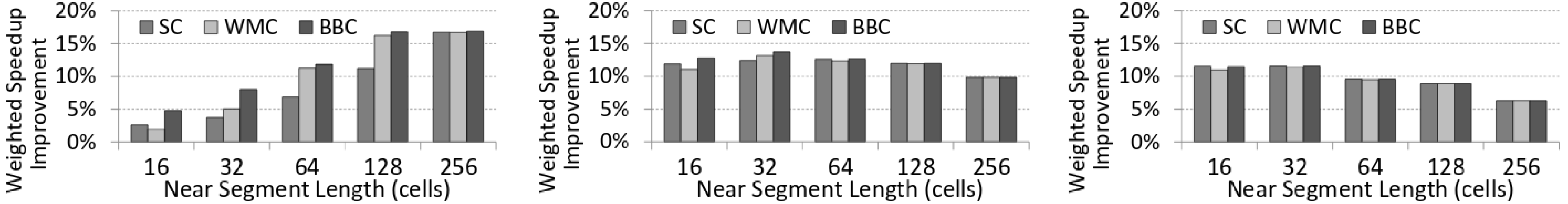


Figure 14. Single-core: IPC improvement, LLC MPKI, Fraction of accesses serviced at row buffer/near segment/far segment, Power consumption



(a) Sensitive - Sensitive **(b) Sensitive - Non-Sensitive** **(c) Non-sensitive - Non-Sensitive**

Figure 17. System Performance: 2-core, Inclusive Cache

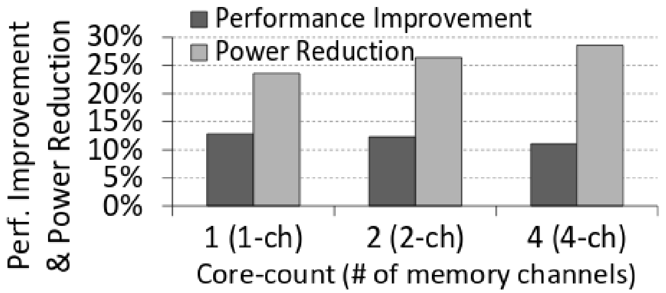


Figure 18. Inclusive Cache Analysis (BBC)

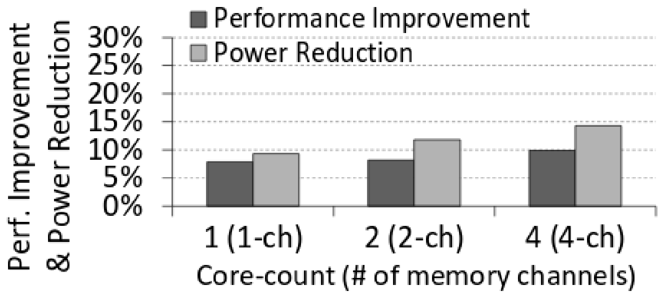


Figure 19. Exclusive Cache Analysis (WMC)

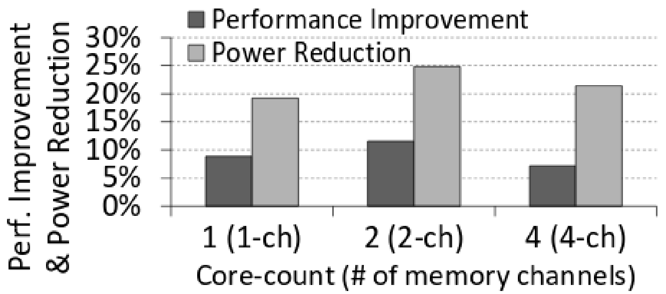
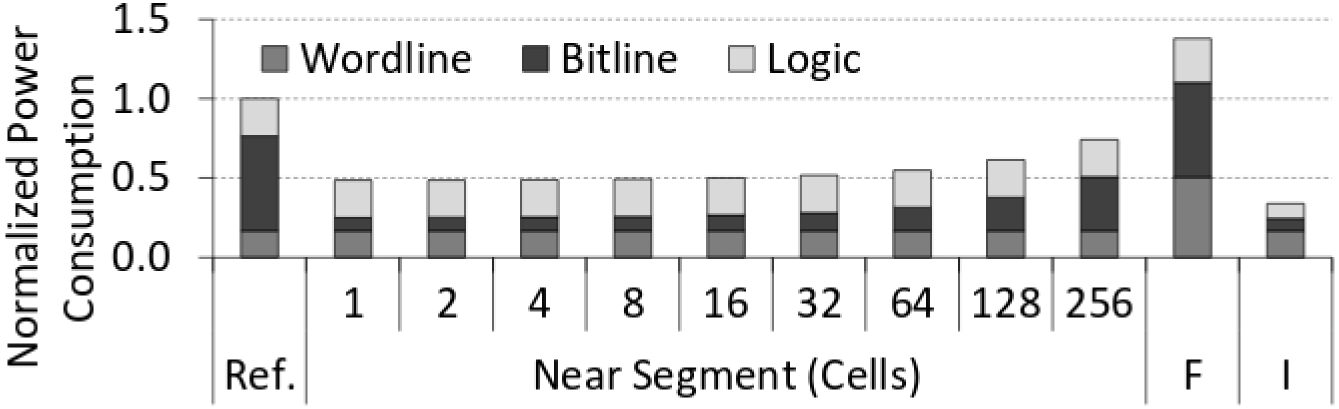


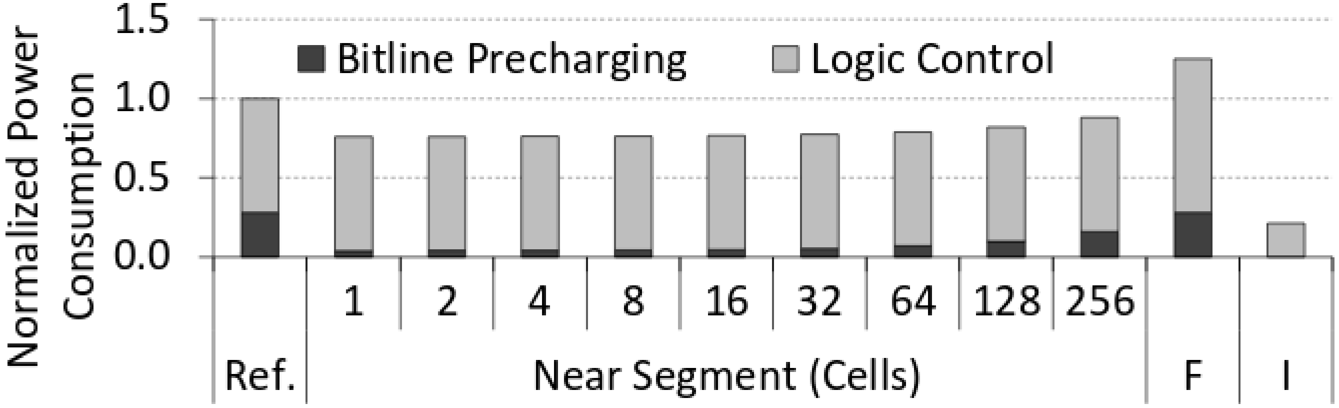
Figure 20. Profile-Based Page Mapping

Power analysis

- Reduced bitline capacitance in near segment → decrease of power usage while accessing data.
- Need to charge transistors while accessing far segment → increase of power usage while accessing the data



(a) Power Consumption for ACTIVATE



(b) Power Consumption for PRECHARGE

† Ref.: Long Bitline, F: Far Segment, I: Inter-Segment Data Transfer

Figure 13. Power Consumption Analysis

More tiers?

- Yo, I’ve head you like cache. So I put cache memory on top of your cache memory.
- Authors considered three “layers” with 32|224|256 cells each
- TRCD = 55%/70%/104%
- TRC = 44%/77%/157%
- Adding more layers costs 3.15% of substrate, and increases power usage of further layers.