ONE LAST LECTURE ON ROUTING

I have a magic trick too! (Nerd magic)...

First, I need to define a <u>squish</u> message pattern.

Given a ~~set~~ subset of processors $\{s_0, \ldots, s_{n-1}\}$ with $0 \le s_0 < s_1 < \cdots < s_{n-1} < P$

Processor s_i sends a message to processor i.

E.g.

source processor		dest pr.
0		0
1	$s_0 \to$	0
2	$s_1 \to$	1
3		2
4	$s_2 \to$	2
5		

Why would I be interested in squish?

Could use it to pair up free + busy processors

Proc	free	busy
F	f_0	
B		b_0
B		
F	f_1	b_1
B		
F	f_2	b_2

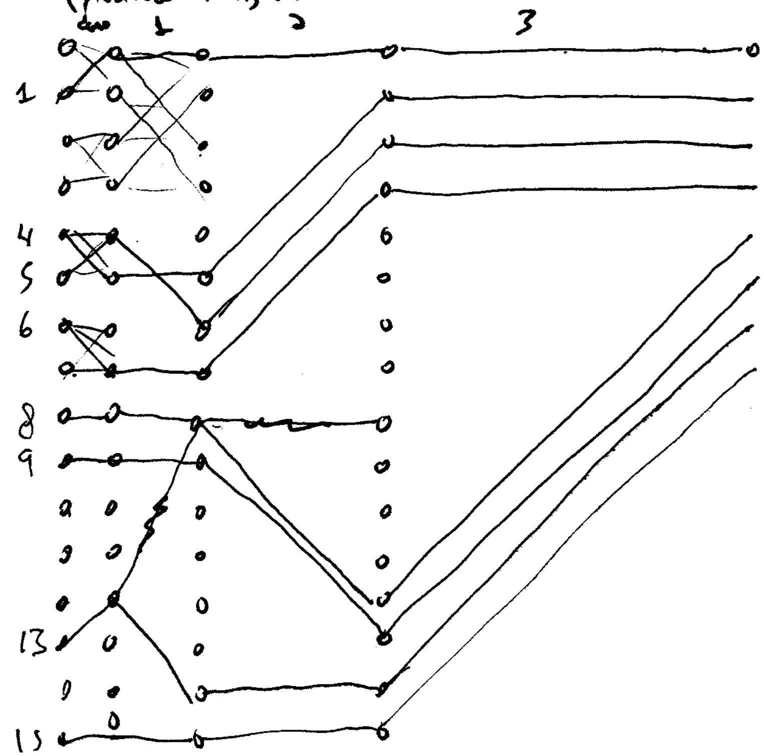
Processor	State		rendezvous	
0	F_0	$F_0 \to$	0	$\leftarrow B_0$
1	B_0	$F_1 \to$	1	$\leftarrow B_1$
2	B_1	$F_2 \to$	2	$\leftarrow B_2$
3	F_1			
4	B_2			
5	F_2			

Now processor 1 knows the address of 1 free + 1 busy processor.

Could be used in ~~a~~ a cilk implementation for example.
Or parallel "cons"

Now the magic: I have a butterfly

Now the Magic! Here is a butterfly (predraw a big one)



Pick some subset that I will squash e.g. 1, 4, 5, 6, 8, 9, 13, 15

$s_0 = 1$ $s_3 = 6$ $s_6 = 13$ 1101
$s_1 = 4$ $s_4 = 8$ $s_7 = 15$ 0110
$s_2 = 5$ $s_5 = 9$ 1011

Look MA, no collisions!

~~This always~~

Thm: This magic always works

Proof: Define: a semicontraction is a 1-1 routing from $\{s_0, \ldots, s_n\}$ to $\{d_0, \ldots, d_n\}$

with $s_i \to d_i$ such that

$\forall i,j$ $$|s_i - s_j| \geq |d_i - d_j|$$

this is subtraction, not hamming distance.

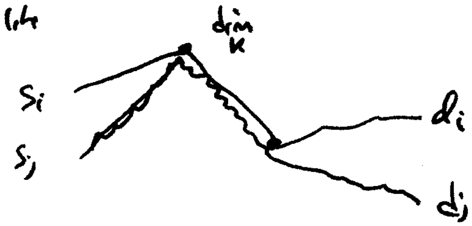
Lemma: Semicontraction routes on butterfly w/o conflict.

proof by contradiction:

Suppose $\exists$ conflict. $\Rightarrow$ some pair must conflict

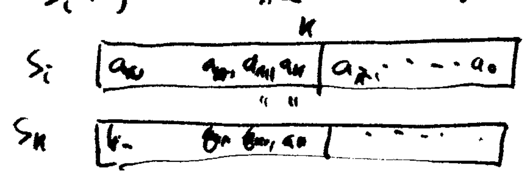
$s_i \to d_i$ conflicts with $s_j \to d_j$

Looks like



Observe $|s_i - s_j| < 2^k$

why? s_i + s_j must agree on ~~bits~~ high bit



but $a_i = b_i$ for $i \geq k$

so $|s_i - s_j| = \left|\sum a_i 2^i - \sum b_i 2^i\right|$

$$= \left| \sum_{i=0}^{k-1} (a_i - b_i)\cdot 2^i + \sum_{i=k}^{n} (a_i - b_i) 2^i \right|$$

$$= \left| \sum_{i=0}^{k-1} (a_i - b_i)\cdot 2^i + 0 \right|$$

$$\leq \left| 2^{k} - 1 \right| < 2^k$$

observe $|d_i - d_j| > 2^k$

Must agree on low bits

			k	
d_i	...	c_{k+1}	c_k	c_0
d_j	..	e_{k+1}	e_k	e_0

$c_i = e_i$ if $i \leq k$

$$|d_i - d_j| = \left| \sum (c_i - e_i) 2^i \right|$$

$$= \left| \sum_{i=0}^{k} (c_i - e_i) 2^i + \sum_{i=k+1}^{n} (c_i - e_i) 2^i \right|$$

$$= \left| 0 + \sum_{i=k+1}^{n} (c_i - e_i) 2^i \right|$$

Some bit must be different, so $> 2^k$.

$\Rightarrow$ not a semicontraction $\not=$ $\boxtimes$.

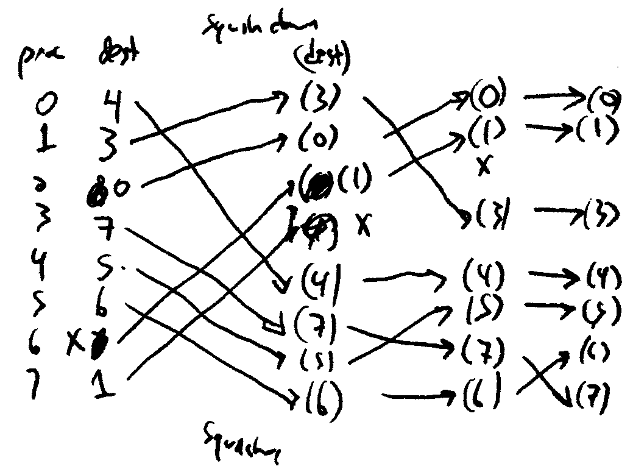
Observe that squash is a semicontraction and $\therefore$ The magic works $\boxtimes$.

What can we do with this magic?
How do we implement it?

Application of magic: routing 1-1 message traffic

Algorithm:

~~pick all.~~
squash down all messages with 0 in bit $(n-1)$.
squash up all messages with 1 in bit $n-1$
+ recurse



How to implement SQUISH.

The problem is to compute the enumeration of the set $\{S, \ldots\}$

let $v_i = \begin{cases} 0 & \text{if } i \notin S \\ 1 & \text{if } i \in S \end{cases}$

i.e.

		$\vec{v}$	w
0		0	
1	s_0	1	
2	s_1	1	
3		0	
4	s_2	1	
5		0	

Want to compute the prefix sum w — also scan +

w_i = the number of 1's in v before i

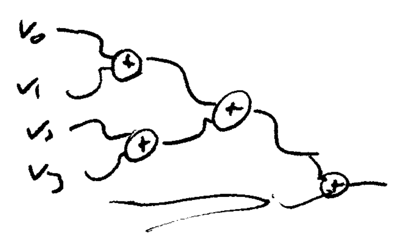
$$w_i = \sum_{j=0}^{i-1} v_i$$

Can easily compute $\vec{w}$ in work P and $T_\infty = P$

Can we do it faster?

Yes. For example we can compute

$$w_p = v_0 + v_1 + \cdots + v_{p-1}$$
$$= ((v_0+v_1)+(v_2+v_3))+((v_4+s_5)+\cdots)$$

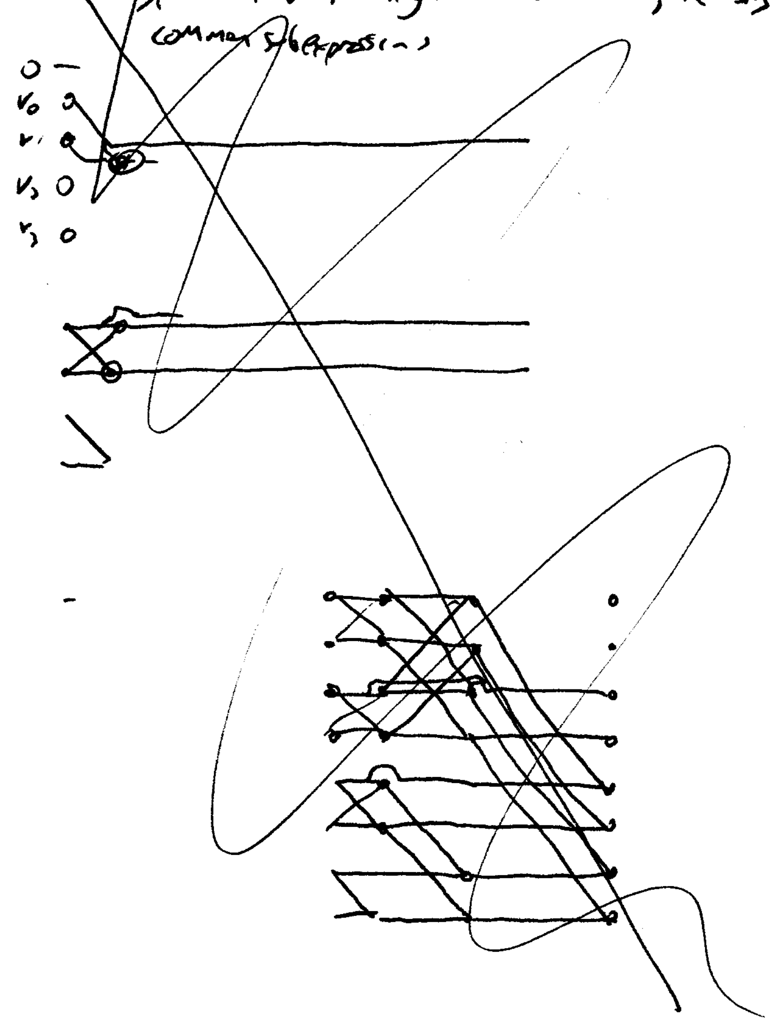


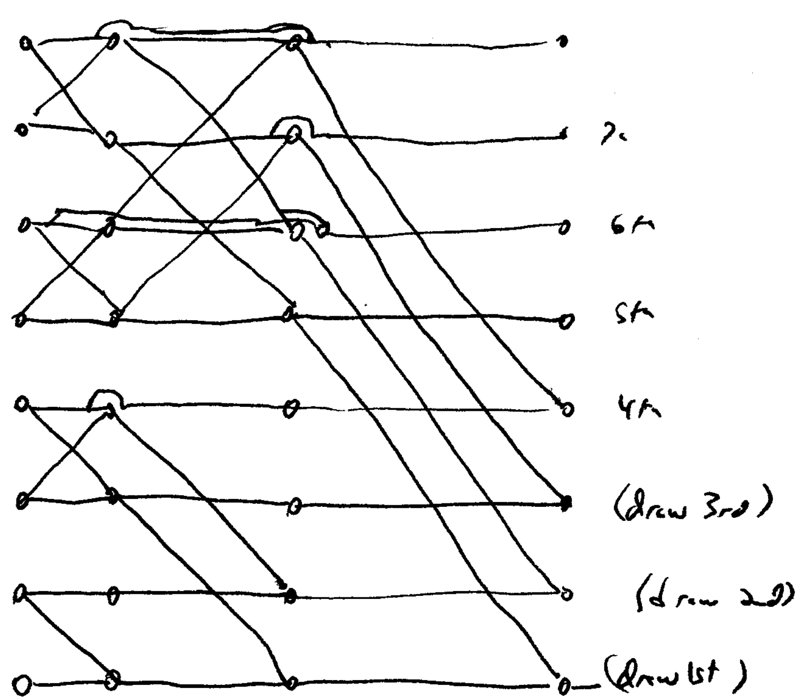
$T_\infty = \lg n$

but work?

On a ~~hypercube~~ Butterfly! — just do it, easy

common subexpressions





It fits on a butterfly!

Analysis: Time to do scan = depth of graph = $\lg n$

how long to add 2 numbers?

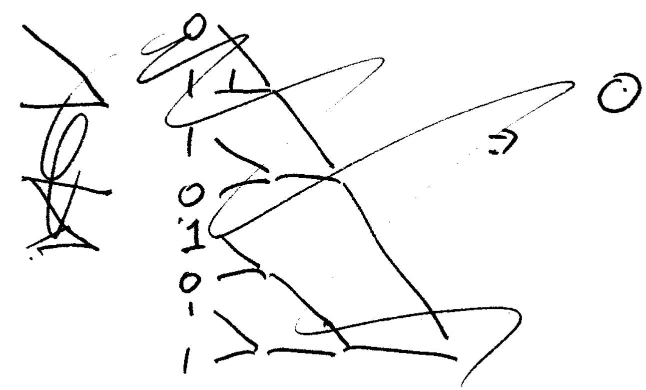
we assume the wire is $O(1)$ bits/time unit

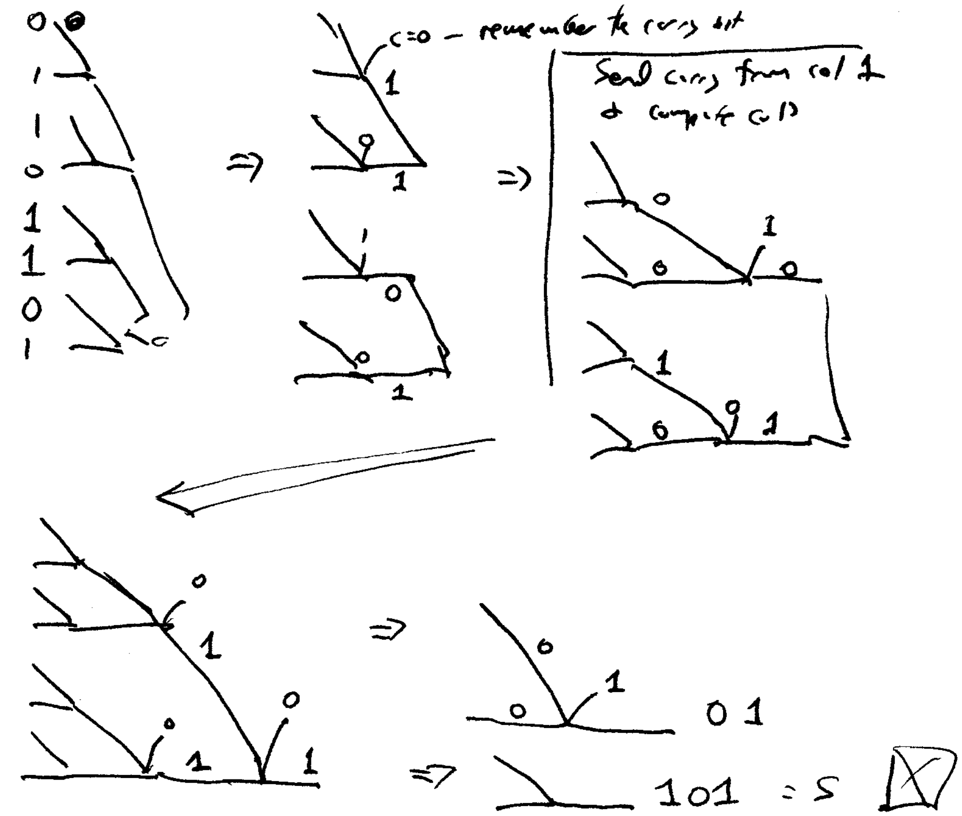
$\Rightarrow$ it takes $O(\lg n)$ time just to send the final answer out the last stage.

~~But, the adds can be pipelined!~~

so is it $O(\lg^2 n)$ time?

No! Bits can be pipelined.





∴ With pipelining get $O(\lg n)$ for /+

~~Total time to run a 1-1 routing~~

$\cancel{t_n = \Theta(\lg n) + t_{n/2}}$

do 1-1 routing on n processors:

Compute scan + up on 0's $\Theta(\lg n)$

Send message in squash down $M + \lg n$ ← message size

compute scan + down on 0's $\Theta(\lg n)$

Send message in squash up $M + \lg n$

do 1-1 routing on two subcubes in parallel

$T_n = 2M + \Theta(\lg n) + T_{n/2}$

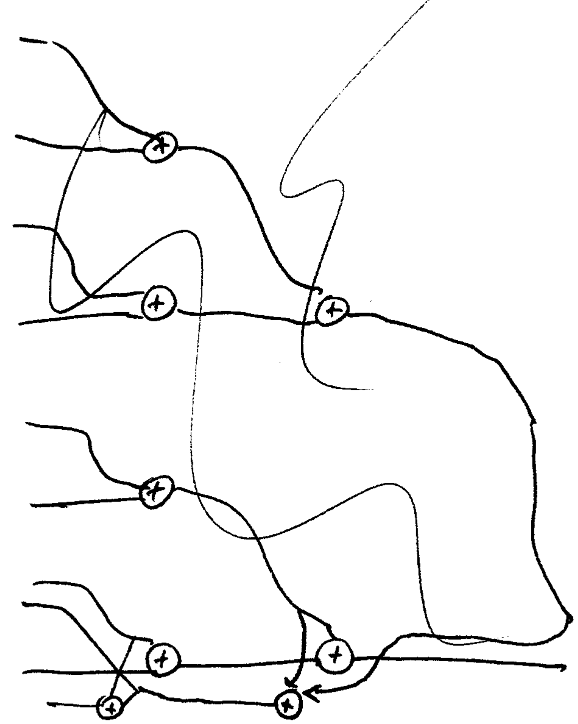
$M = \Omega(\lg n)$

$\Theta(M) = \lg n$

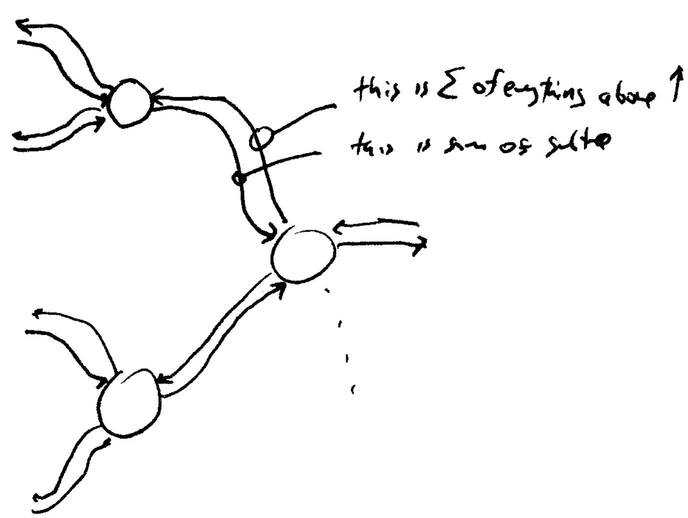
$T_n = O(M) + T_{n/2}$

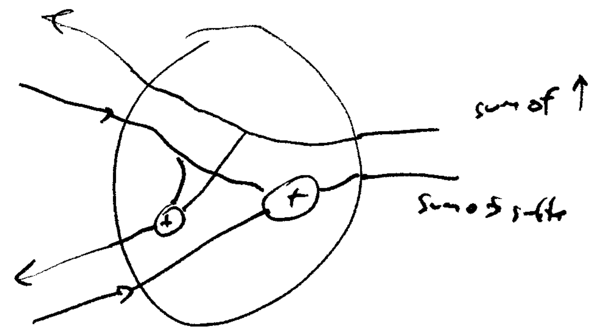
$= O(M \lg n)$

Another way to do parallel prefix: with two trees
superimpose

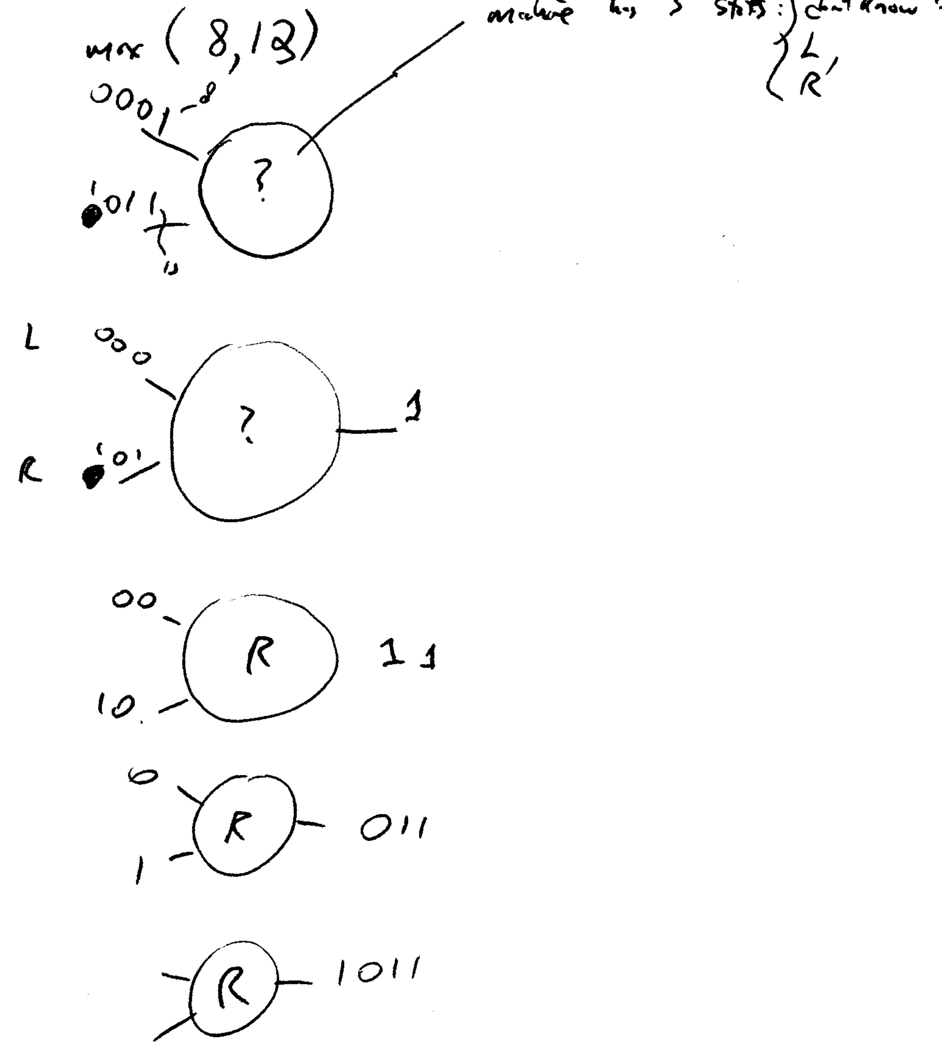


Another way to do /+
with depth 2lg P adders
but work only P
(butterfly is depth lg P, work P lg P)



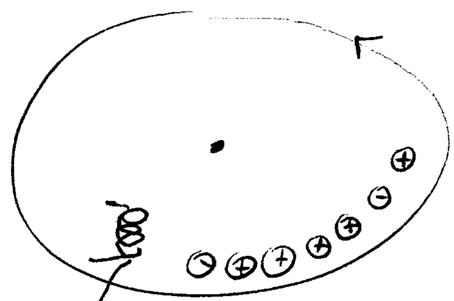


Also can bit-pipeline max; feed in most significant bits first



Disks:

A spinning platter covered with magnets



encode info in the orientation of magnet.

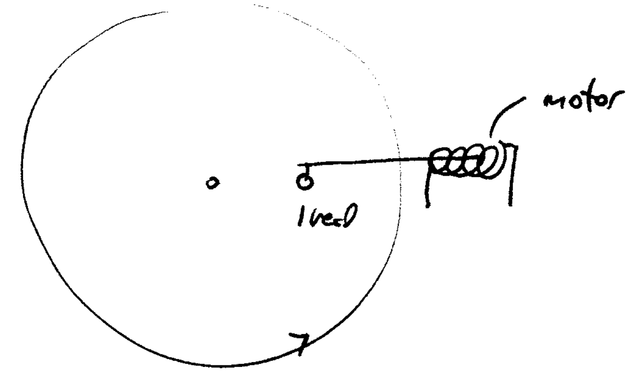
To write: position an electromagnet over a spot and apply current, it switches the state

To read: in principle, ~~the~~ goes in reverse. Sense current from the induced field.

Reality: ~~Gigantic Magneto~~
GMR — Giant Magneto Resistance
a device whose resistance is a fn of magnetic fields. Very sensitive.

GMR discovered in late 80's
⇒ about 10 years to appear in disks.
About 20 Gb/in²

How to position the head?



disk spins
motor moves head radially ("seek")
⇒ can see any spot on disk

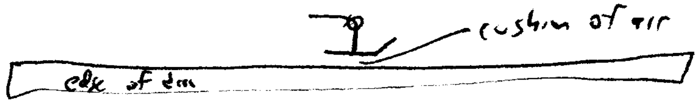
Motor is a linear induction motor

(Old motors were stepper motors. time to seek distance n was $\Theta(n)$
with linear induction motor, head accelerates to ½-way point ⇒ time $\Theta(\sqrt{n})$

today
avg seek time = 4.9ms read, 5.4ms write
best seek time = ~1 ms
worst = ~9 ms
rot = 10,000 rpm (expensive + hot) 6 ms
5,400 rpm 11 ms
(½ that on average.

More data near edge of disk
=> trick: put your data on the outer part of the disk to reduce # of seeks.

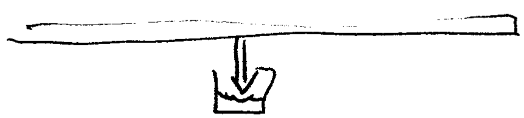
Head "flies" above disk surface



Disk bearings use fluid-dynamics ~~be~~
(not ball bearings - about 2 years ago)

- Quiet
- No runout (vibration, making accurate placement tight)
  Non Repeatable Runout

the disk itself "swims" in



with ball bearings heat + vibration were the disks' big enemy) causing the bearings to deform.

Now - I don't know what causes disks to wear out.

~~Obscure cost to write] reduce but is~~
Drive can move
~~488~~
458-799 Mb/s (up to 200 MB/s)

Since avg seek is 5ms seek
$$\frac{3\,ms}{8\,ms}$$ rot

=> 16 MB/s transfers needed to keep the disk 1/2 busy!