grammars / parsing / compiling

optimization

Chomsky hierarchy

recursively enumerable

↑

context-sensitive

|

context-~~insensitive~~ "free" CFG

|

regexps

(((3)))

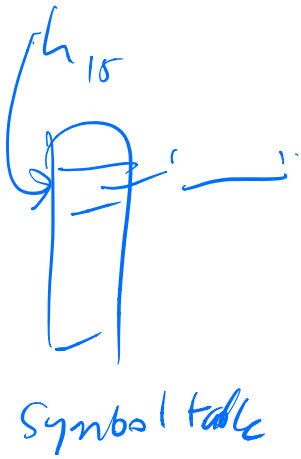
ALGOL-60

Backus-Naur form BNF

expr ::= '(' expr ')' |
number |
expr op expr |
unaryop expr

"var" "variable"

42



backtracking ~ exponential time

N tokens 2^N

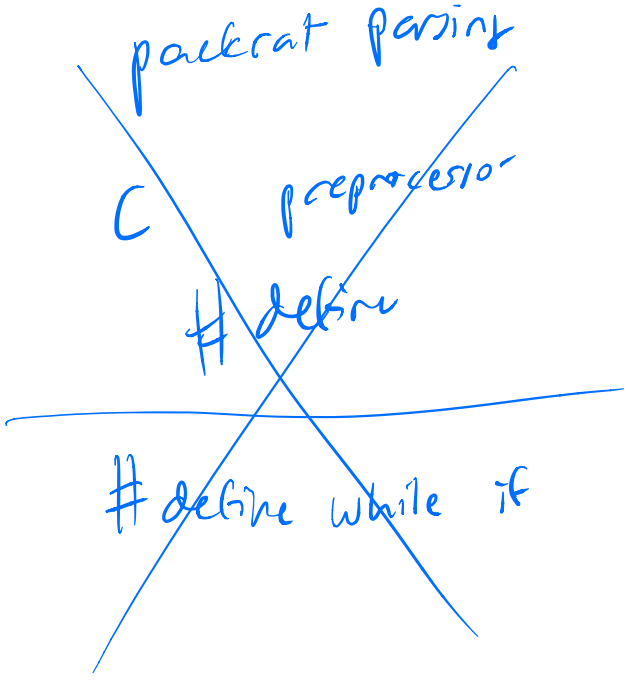
text → lexer → tokens → parser → parse

lex
flex

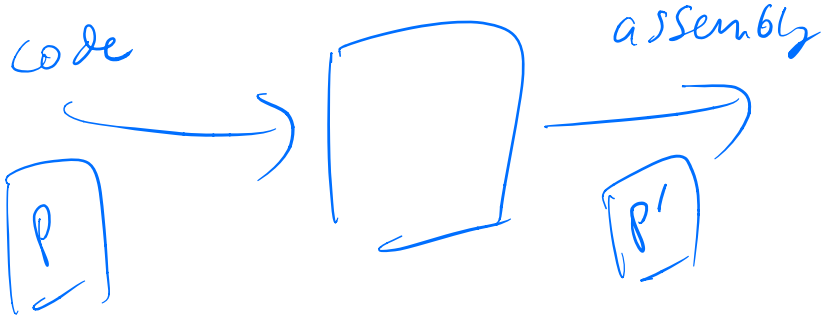
grammar → yacc / bison → parse

Java... Antlr

LL(1)
LALR(1)

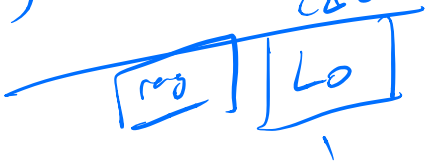


optimization



semantics-preserving

$\forall$ inputs : $P(i) = P'(i)$

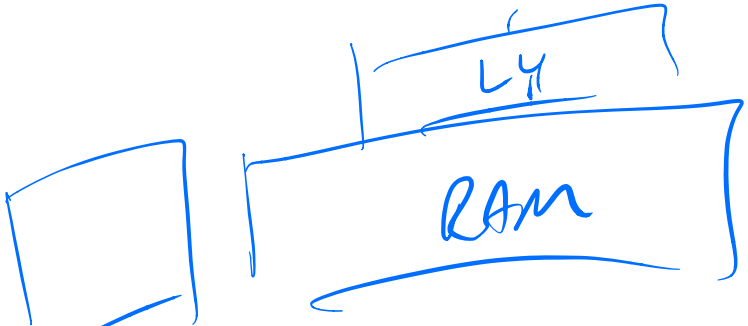


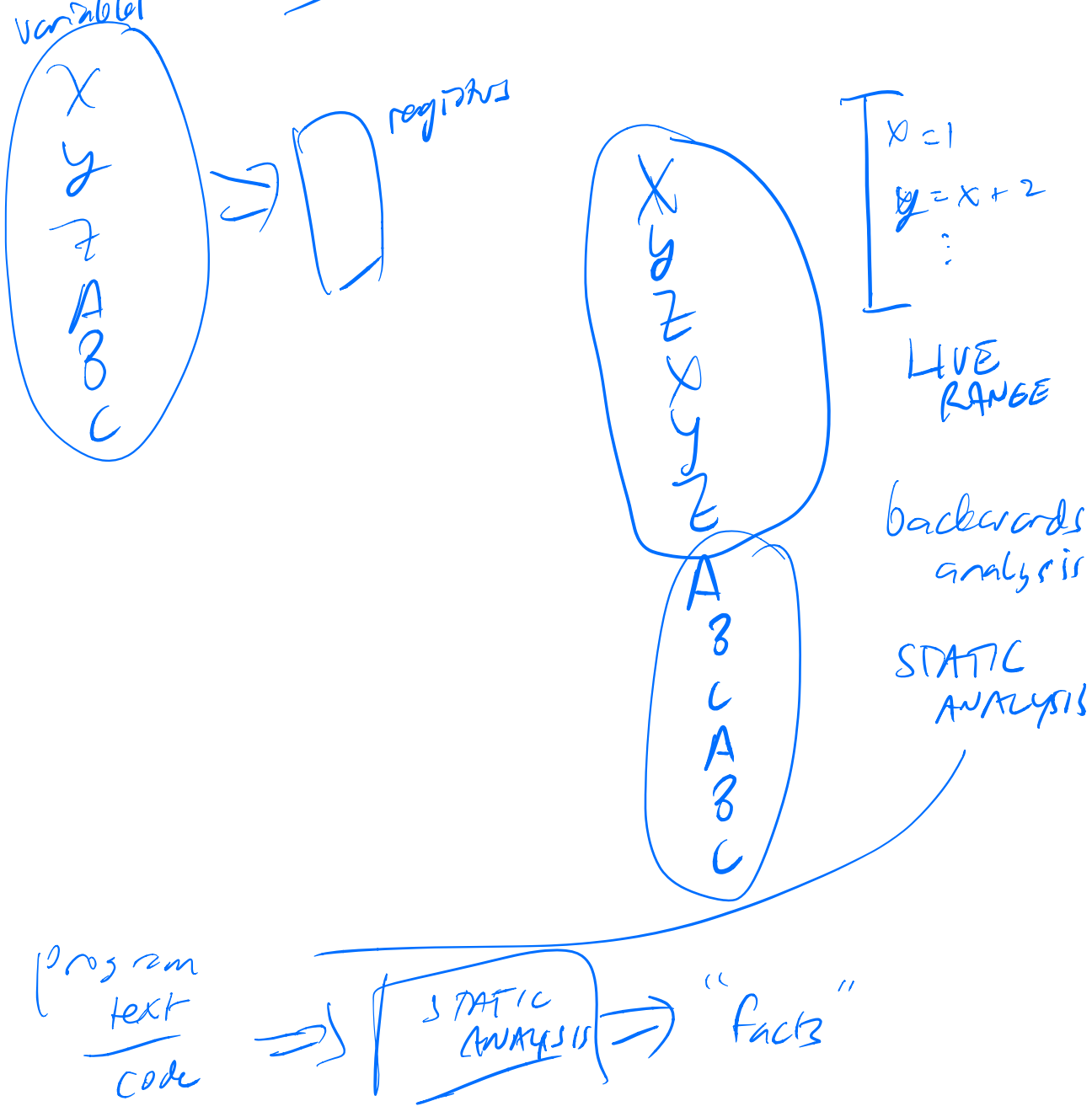
— caches

registers

memory

memory hierarchy

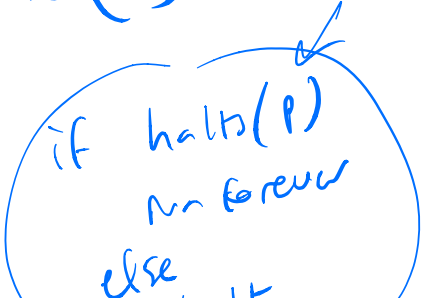




Halting problem

P : ∀ i : input : P(i) terminates

halts(P)



conservative

STRAIGHT LINE CODE

LOOPS

RECURSION

FUNCTION CALLS

FIRST-CLASS FUNCTIONS

EVAL EVIL

POINTERS

FUNCTION

INTRAPROCEDURAL

INTER PROCEDURAL

$x = [1, 2, 3]$

INT * x

$y = [1, 2, 3]$

INT * y

ALIAS ANALYSIS

POINTER ANALYSIS

FOR

x[i]

x + i

y[i]

y + i

f(int* x, int* y)

{

x = A

y = B

if ( ) {

x =

} else {

x = y

}

$x: \{A\}$

$y: \{B\}$

$x_1: \{A, z\}$

$x_2: \{A, y\}$

register allocation
= graph coloring
NP-complete

strength reduction

X = pow(3, 2)

X = 3 * 3

X = 9

3^2

y = 12

z = y + 1 → 13

x = 2 → 13

z = 26

Constant propagation

inlining — exposes optimization opportunities

```
int add(int x, int y) {
  return x + y;
}
```

reduces fn call overhead

```
foo()
  int a = 12;
  int b = 13;
  add(a, b);
```

closed-world

Whole-program

int c = addOne(a, b);
= a + b 25

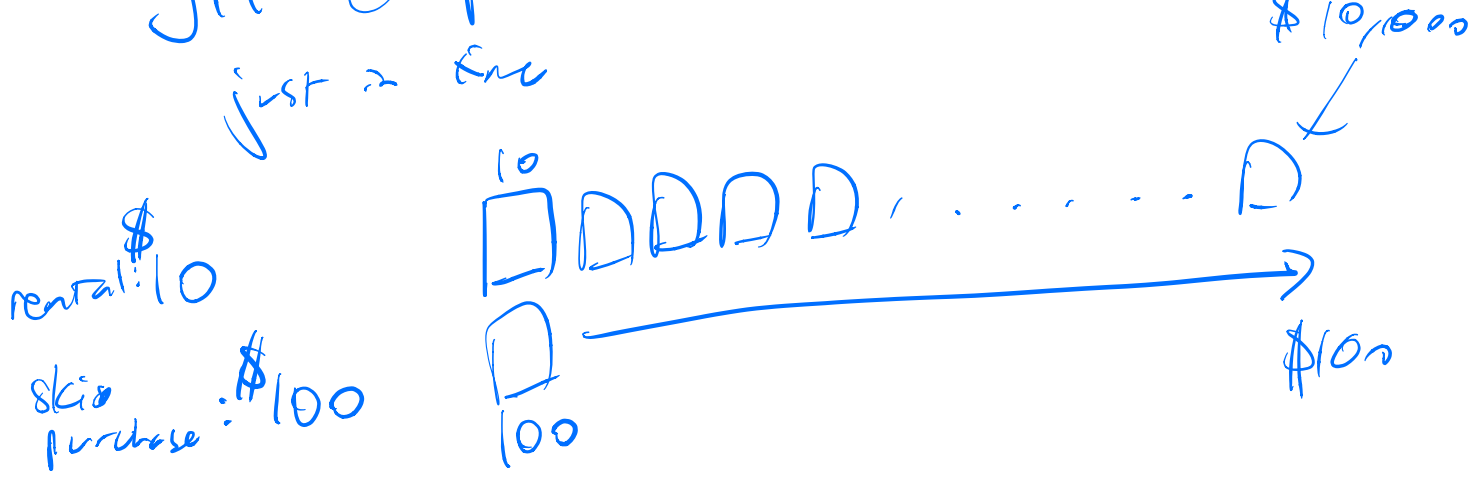
analysis
Modular analyses

FORTRAN - OPTIMIZING COMPILATION

LISP - INTERPETERS

switch (_) {
case ADDONE:
var(p) = var(p) + 2
break;
:

JIT compilers
just in time



Ski rental problem

rent until you spend $100
Then you buy

worst-case: Spend $200
could have spent $100

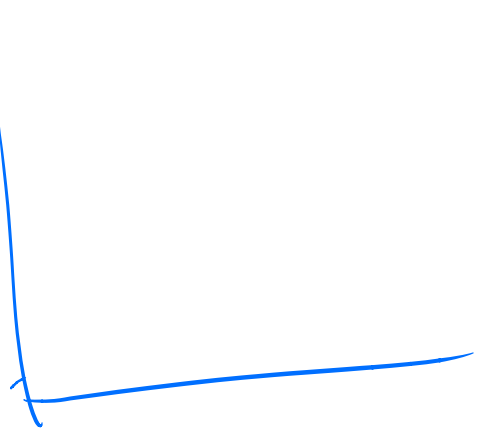
Interp

Fast JIT - -O0
-O1
-O2
⋮

Jim Larus
LARUS

dumb
slow code

smart
fast code



parse
↓
internal representation
↓
AST abstract syntax tree

homoiconicity

LISP — lambda calculus

functional programming

function-centric
functions first-class objects

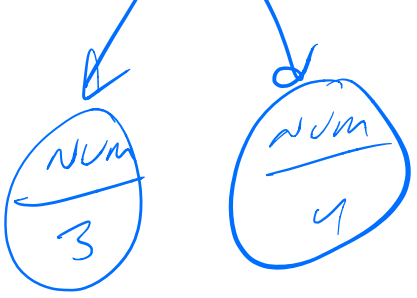
program as data

garbage collection

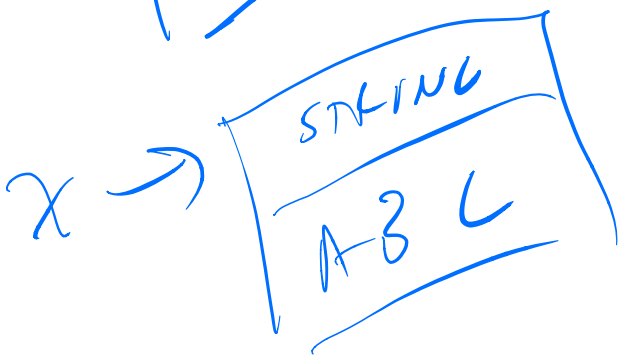
3 + 4

EVAL

dynamically typed prefix



(+ 3 4)



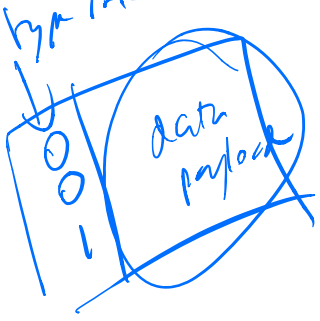
STATIC typed vars.

DYNAMIC typed data

(+ x 1)

boxed values

tagged



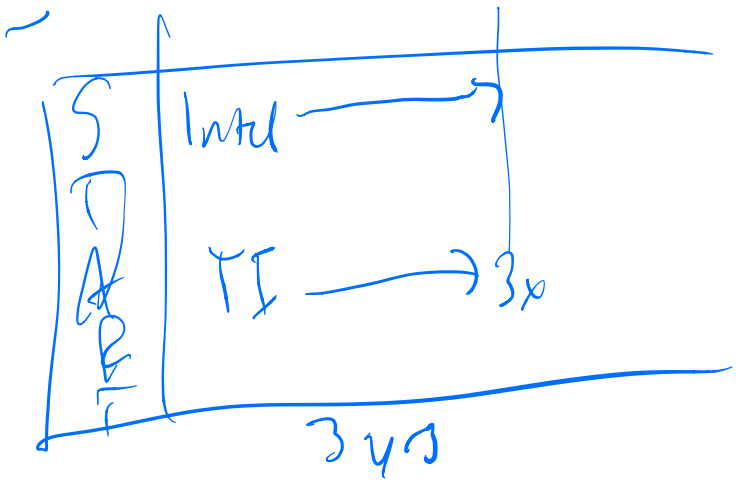
LISP machines 70s-80s

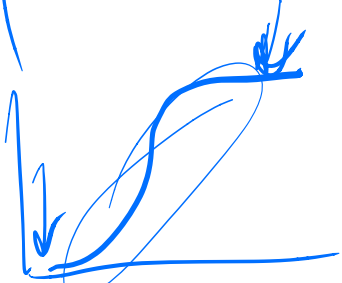
TI Explorer

DB machines

computer vision

neural net





Moore's Law

gates in area doubles ~ 18 mo!

Dennard scaling

↑ density ⇒ ↑ clock speed

Proebsting's Law

compiler opts. double perf every 18 YEARS

Self

Chambers Ungar Hölzle

Jeff Dean

GC

Mark-sweep

reachability

explicit mem mgmt

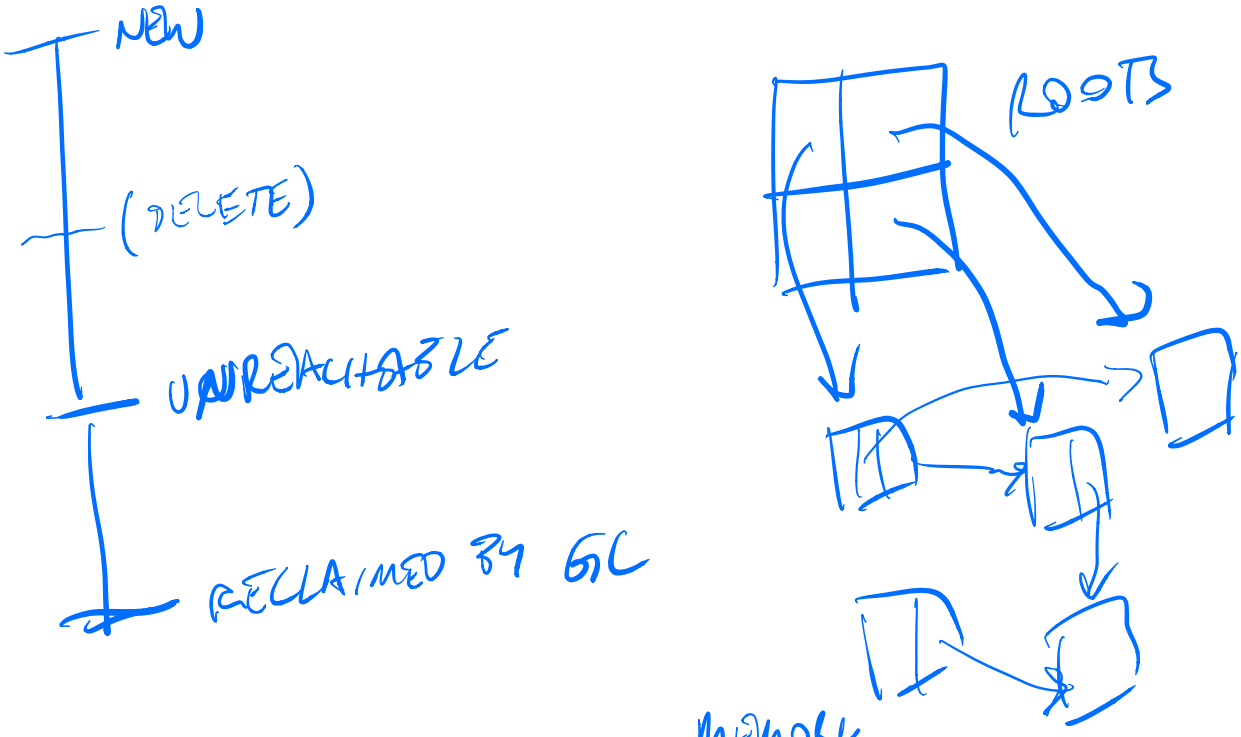
malloc/new

free/delete ∈ object "dead"

live

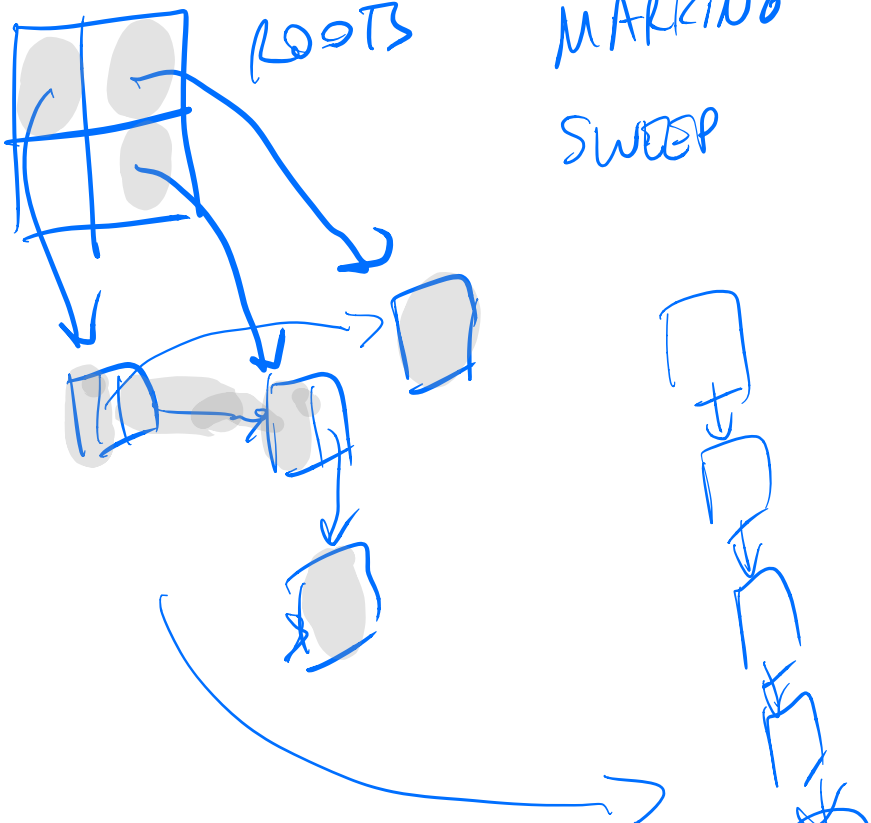
liveness

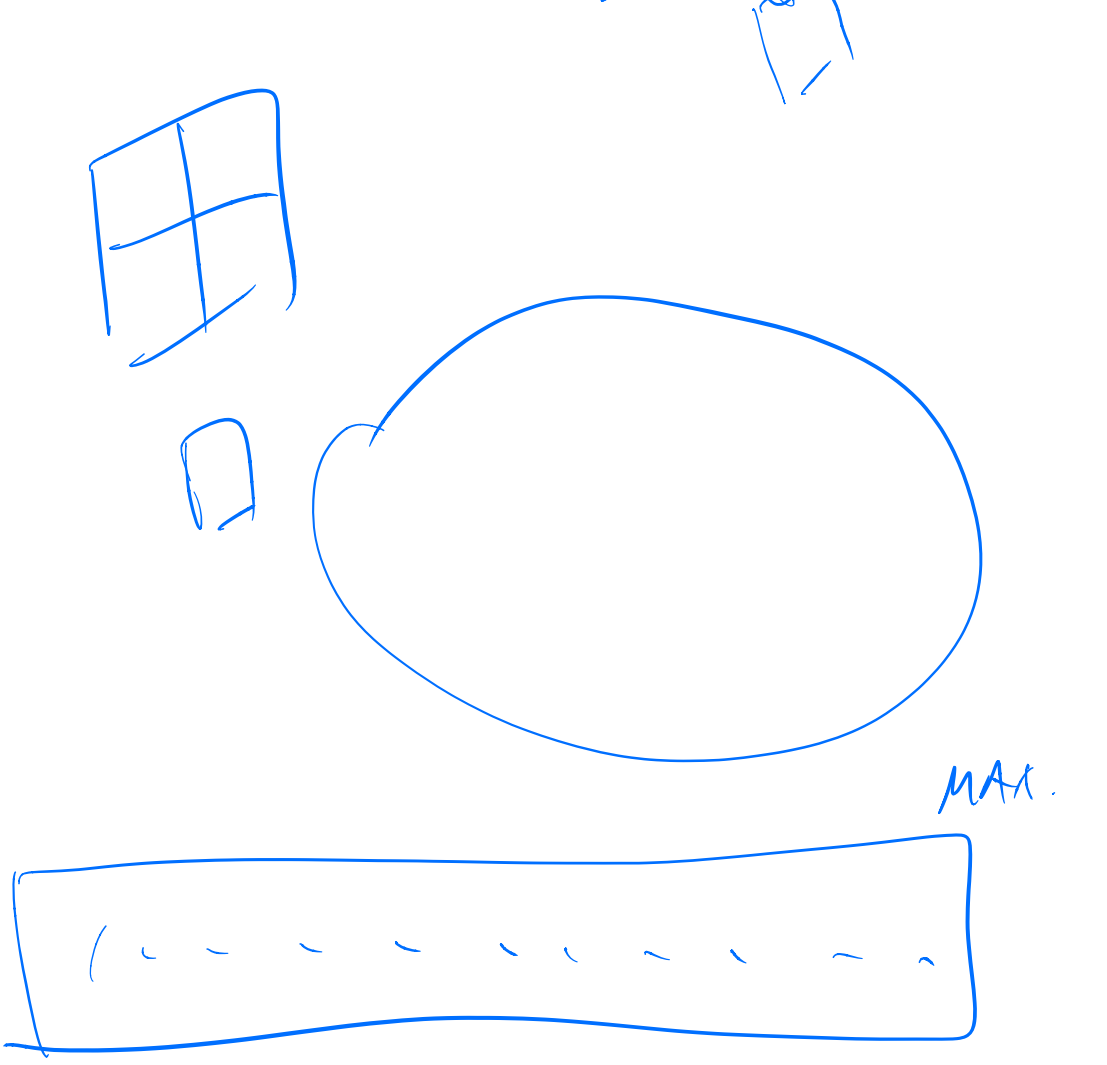
reachability ≈ Oracle

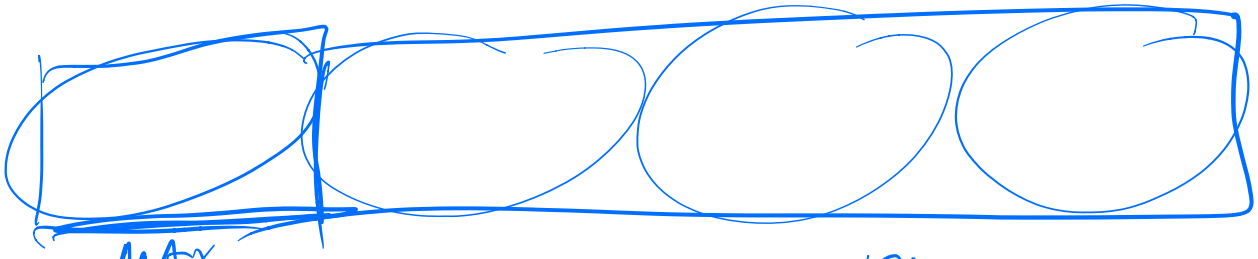


WHITE — never been visited

BLACK — visited




MAX.
GC: STOP THE WORLD
pause time
GC latency
63.9999 GB



Overprovisioning

Quantifying the Perf. Impact of GC vs. Explicit Mem Mgmt

$\sim$ 3x - 5x more RAM $\simeq$ malloc/free

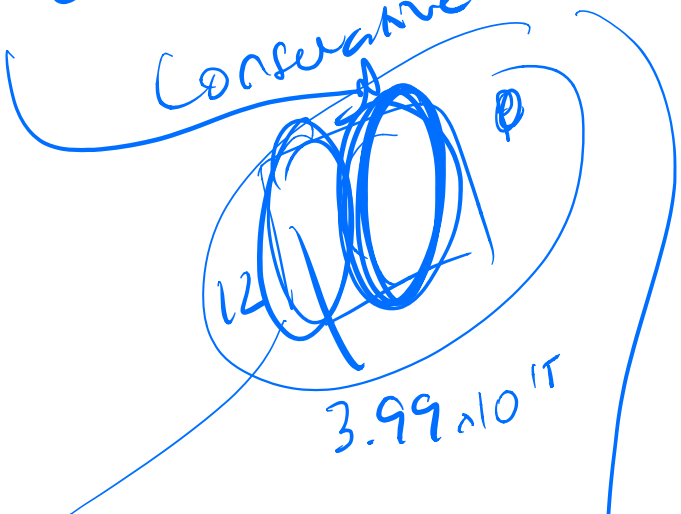
UNCOOPERATIVE GC

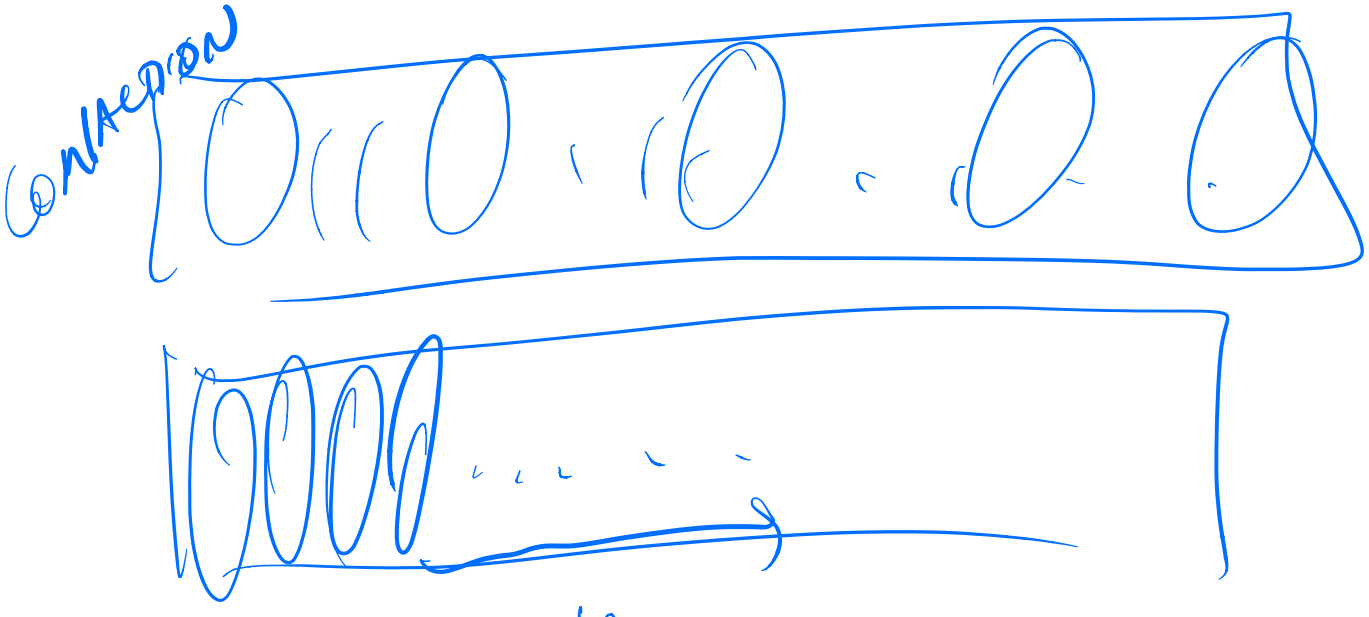
Hans Boehm

duck test

PRECISE GC

Conservative GC





localizing
enhancing

MARK-SWEEP - COMPACT

---

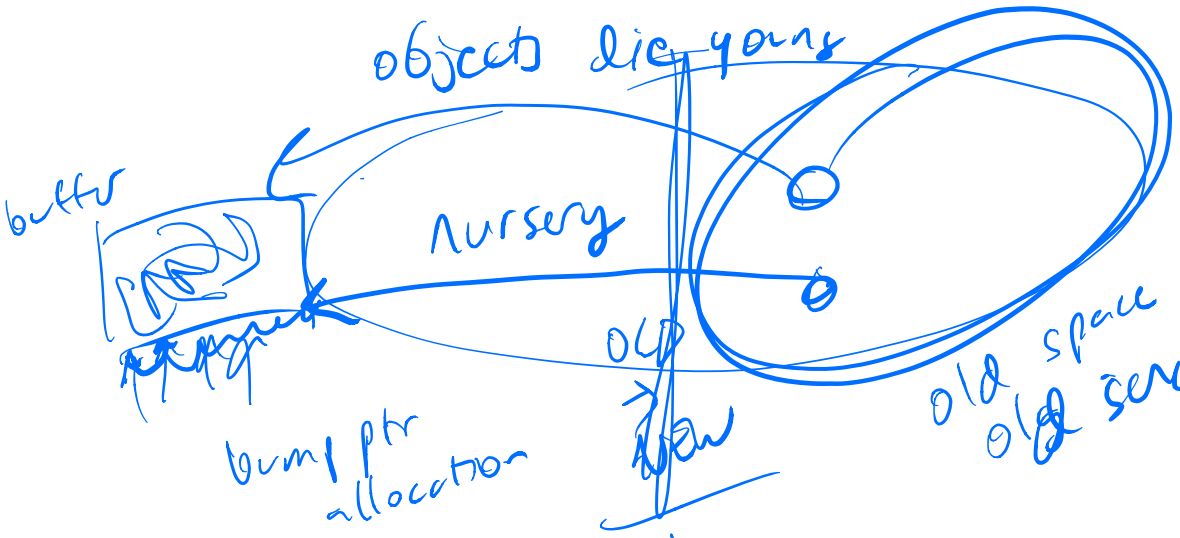
reachability

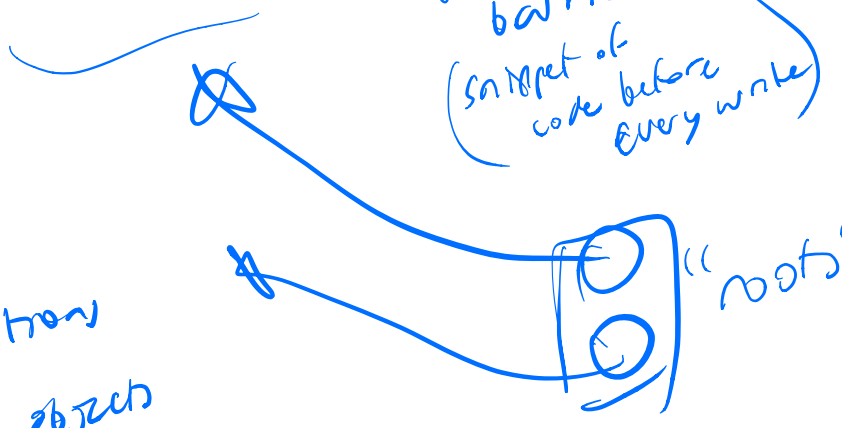
"scanning"

MS
MSC
generational GC

generational hypothesis

objects die young





after allocation

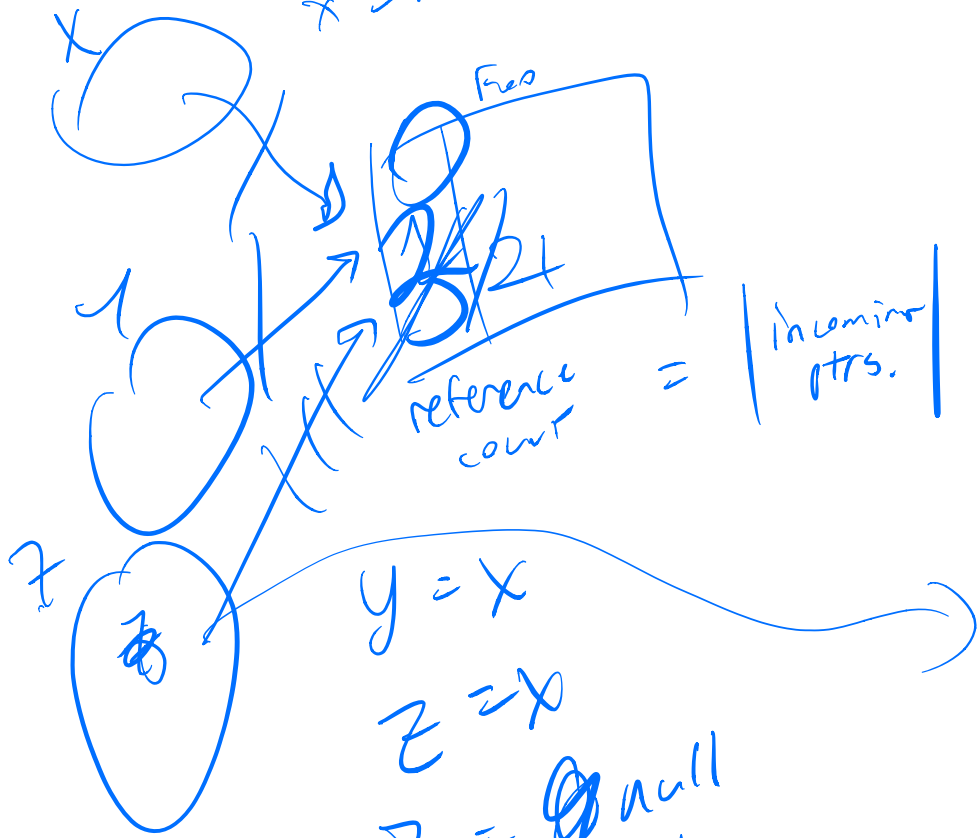
young objects are quickly reclaimed

⇒ delaying full heap GCs

pre-tenuring immortal

reference counting

x = new Foo



ptrupdate (X, q)

if *X = q do nothing

else ref(*X) --

ref(q)++

if (ref(*X) == 0)

delete *X;

C++ smart pointers

cycles

X = null

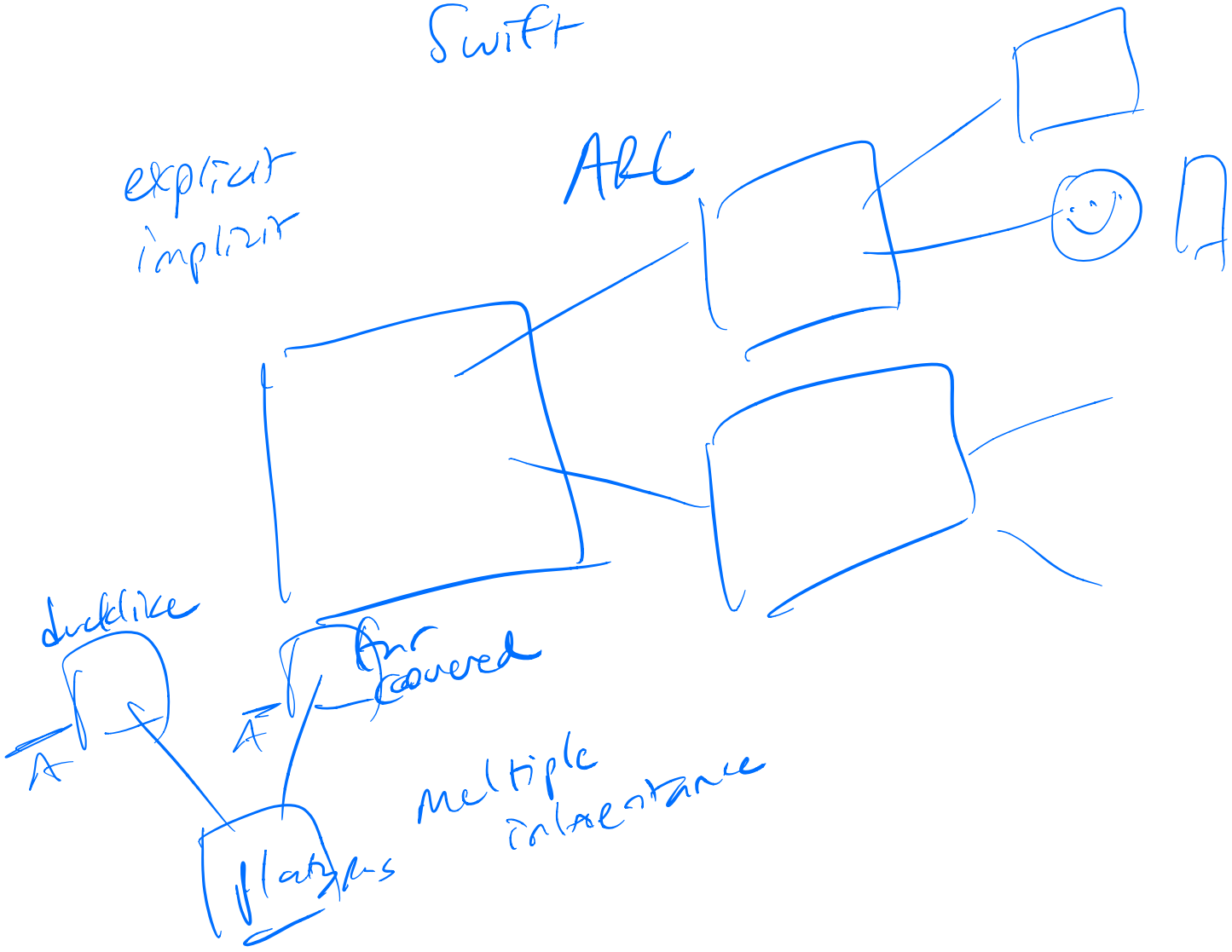
cycle collection
≈ GC

finalizer
RAII

delete o;
class Obj {
~Obj() {
}

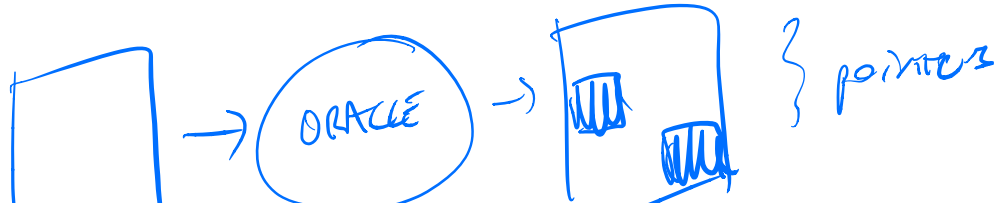
region
pool allocation
arenas

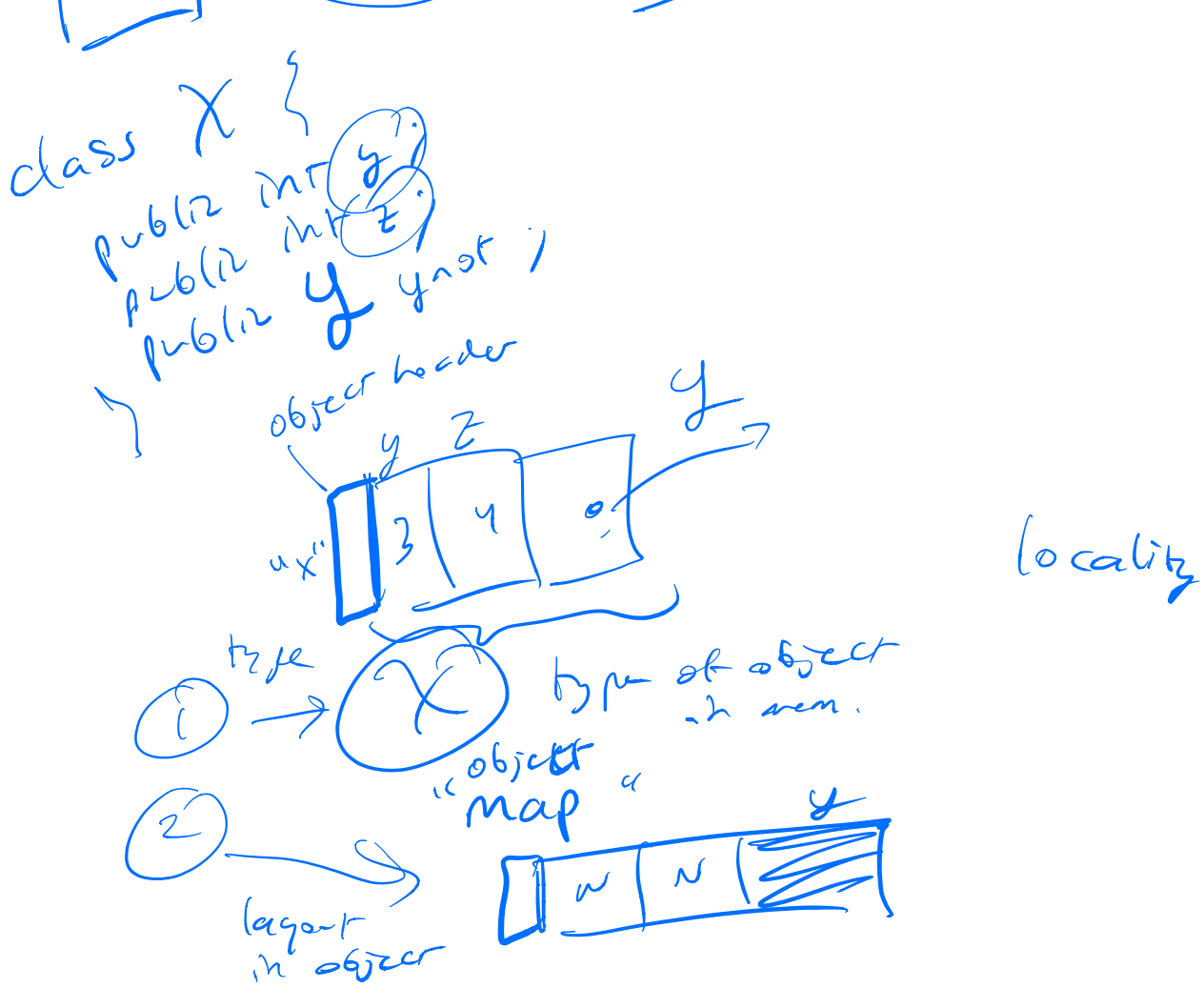
RC → Objective-C
Swift



Boehm collector
conservative GC
— cannot distinguish ptrs. from values

precise GC
— can identify ptrs.





managed languages

garbage collection

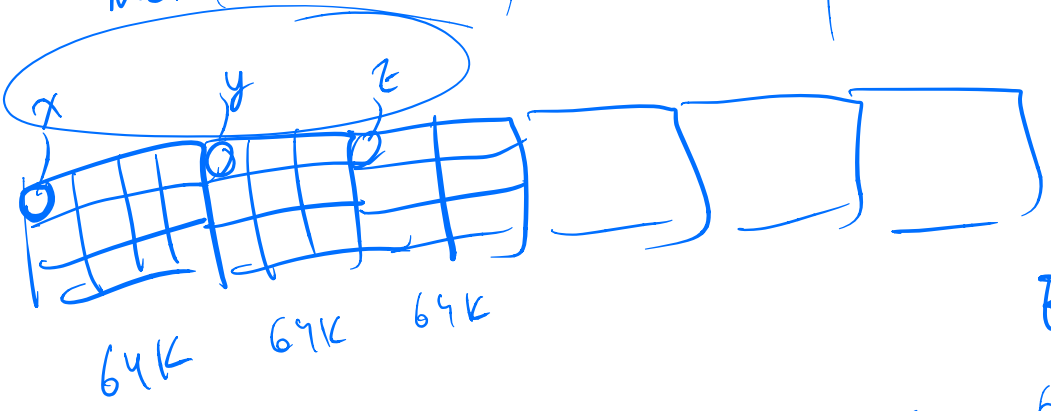
type safety —

bounds checking

errors — uninitialized reads

X, Y, Z, ∝

metadata



BiBOP

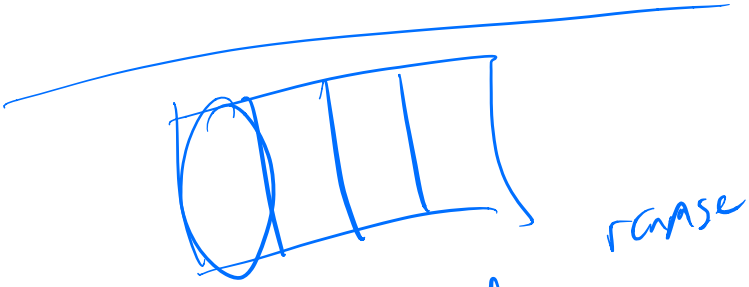
big bag of pages

threaded freelist

Morris worm

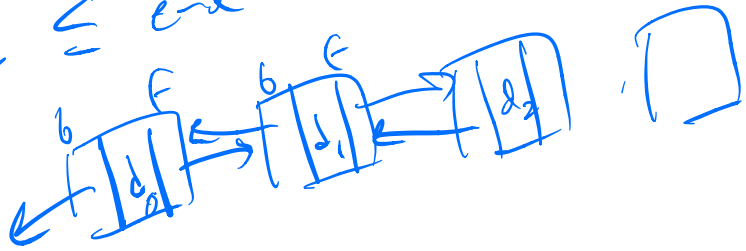
DDoS

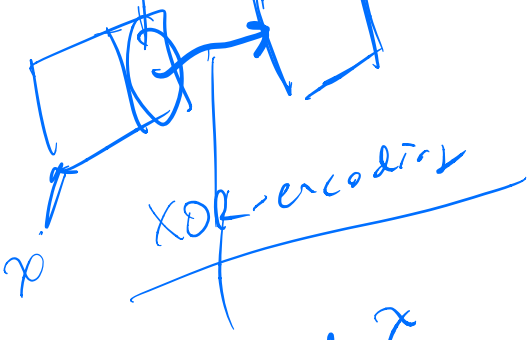
malware

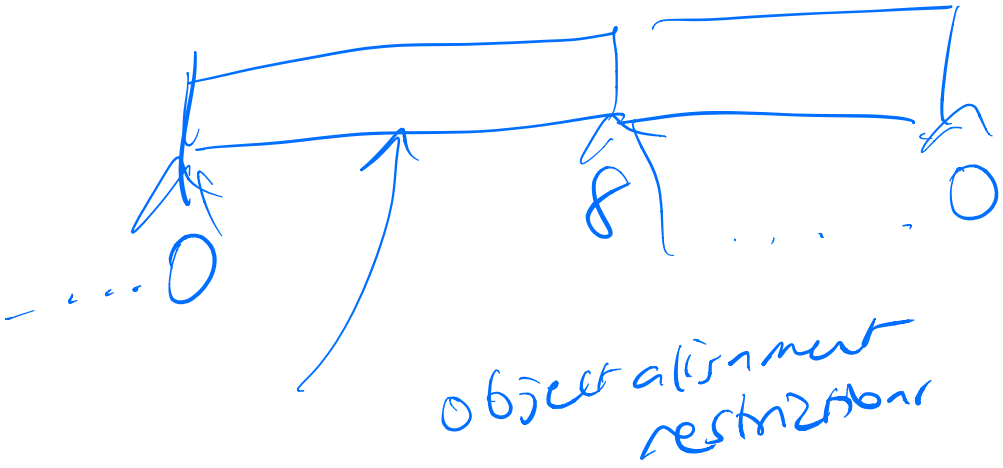


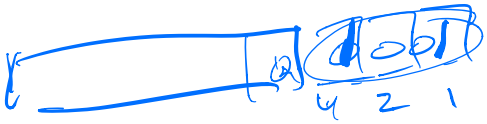
range test

$base \leq x \leq end$

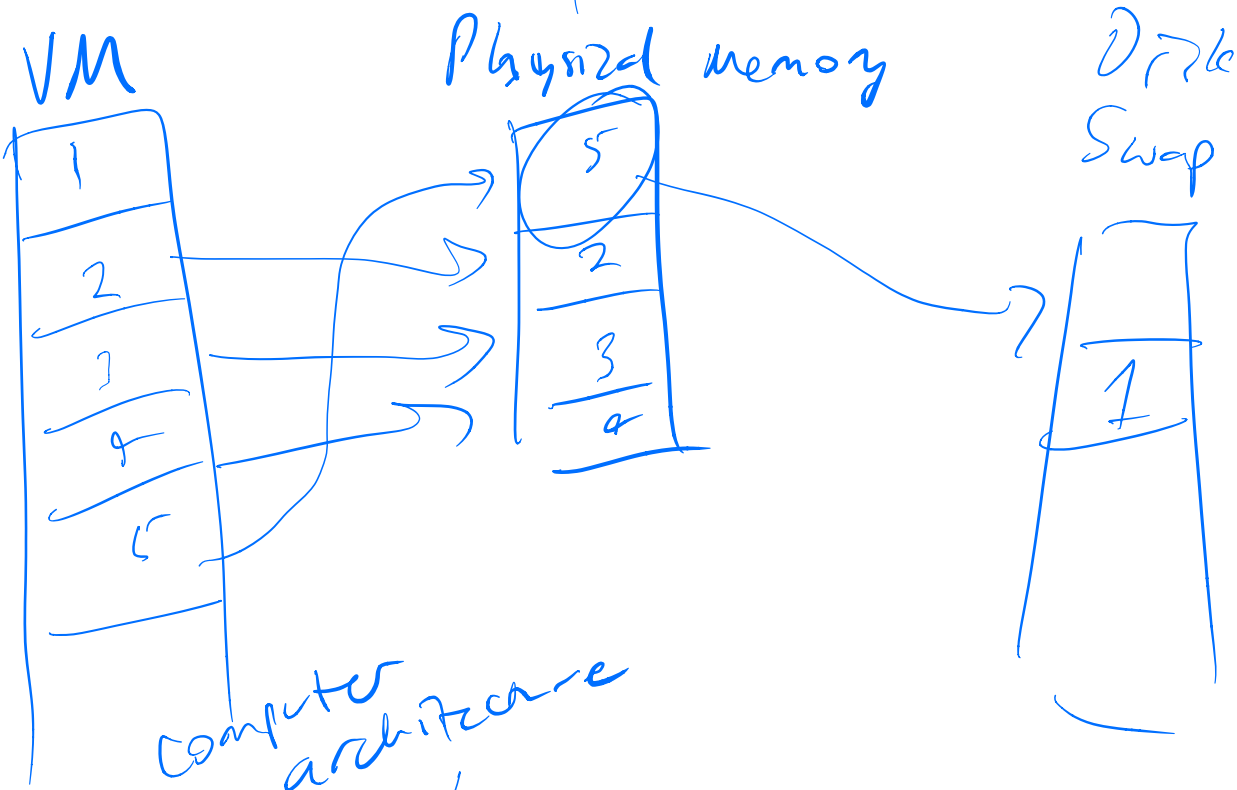








signal means passing



computer architecture

ISA

Instruction Set arch

decoupling

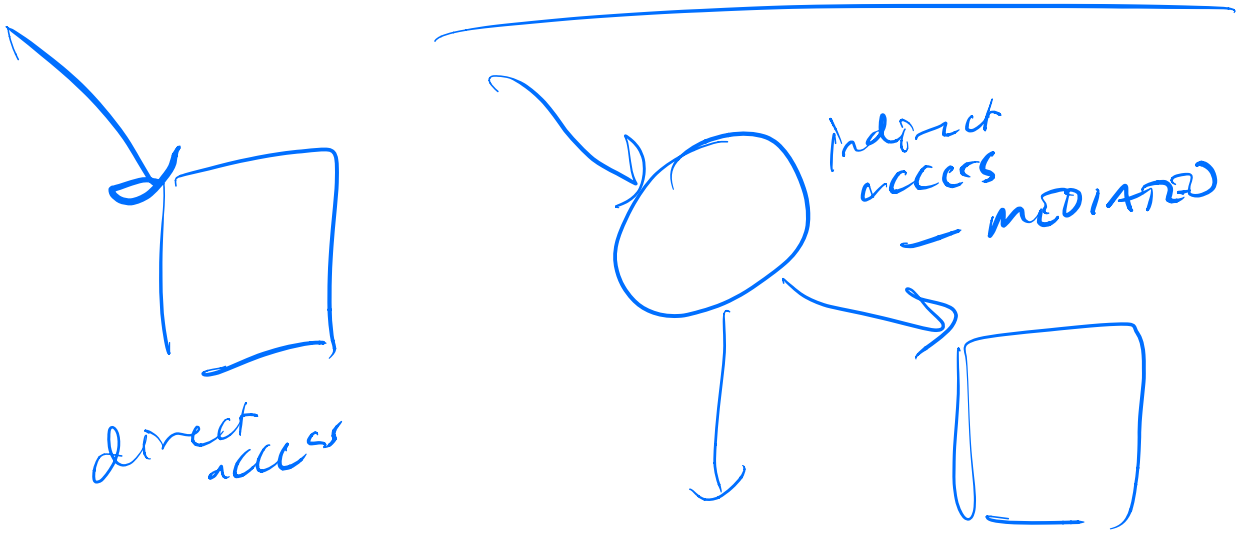
Microarchitecture

virtualization

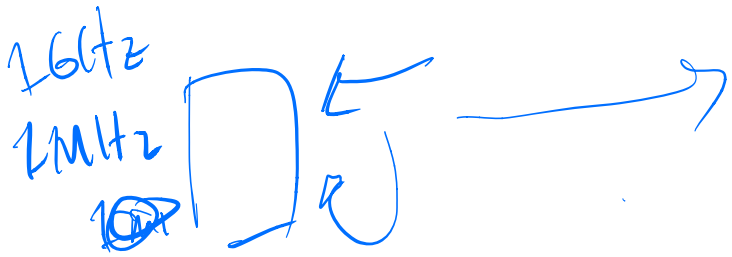
memory hierarchy

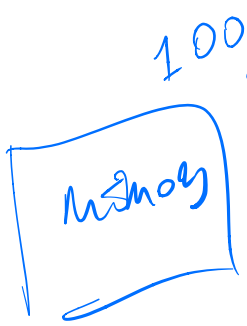
- caches

"all problems in CS
are solvable
by adding
one level of indirection"

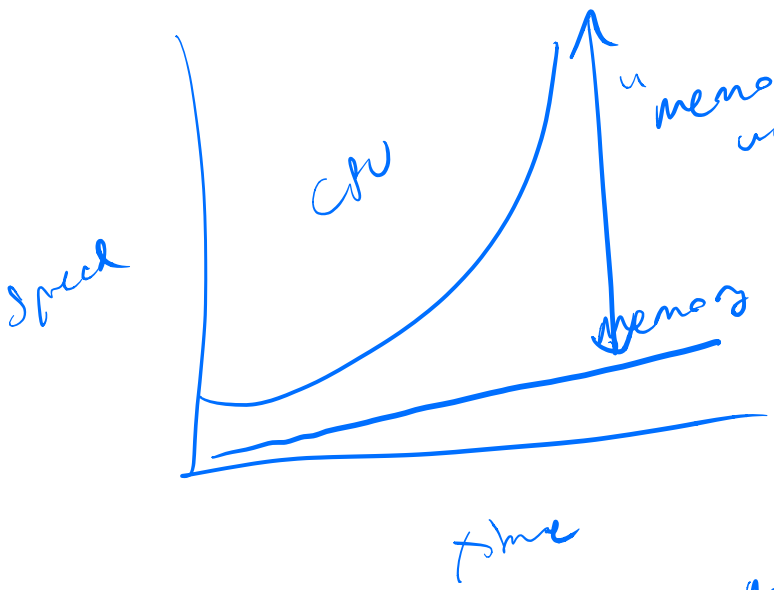


Merdel Rosenblum
Carl Waldspurger



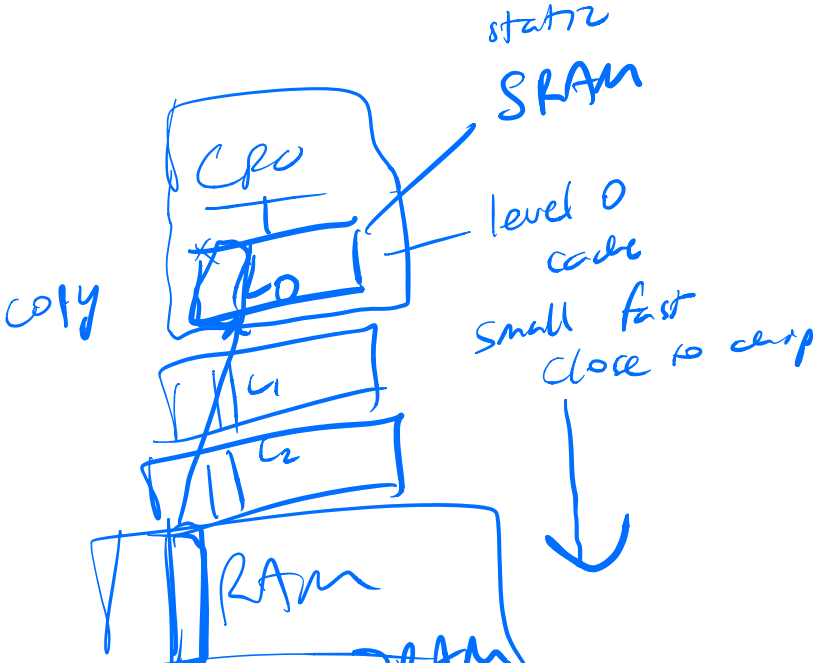


queuing model



CISC grows over RISC solves this

memory hierarchy



copy

ADD MOVE

cache line

326 → 1286 → 2566

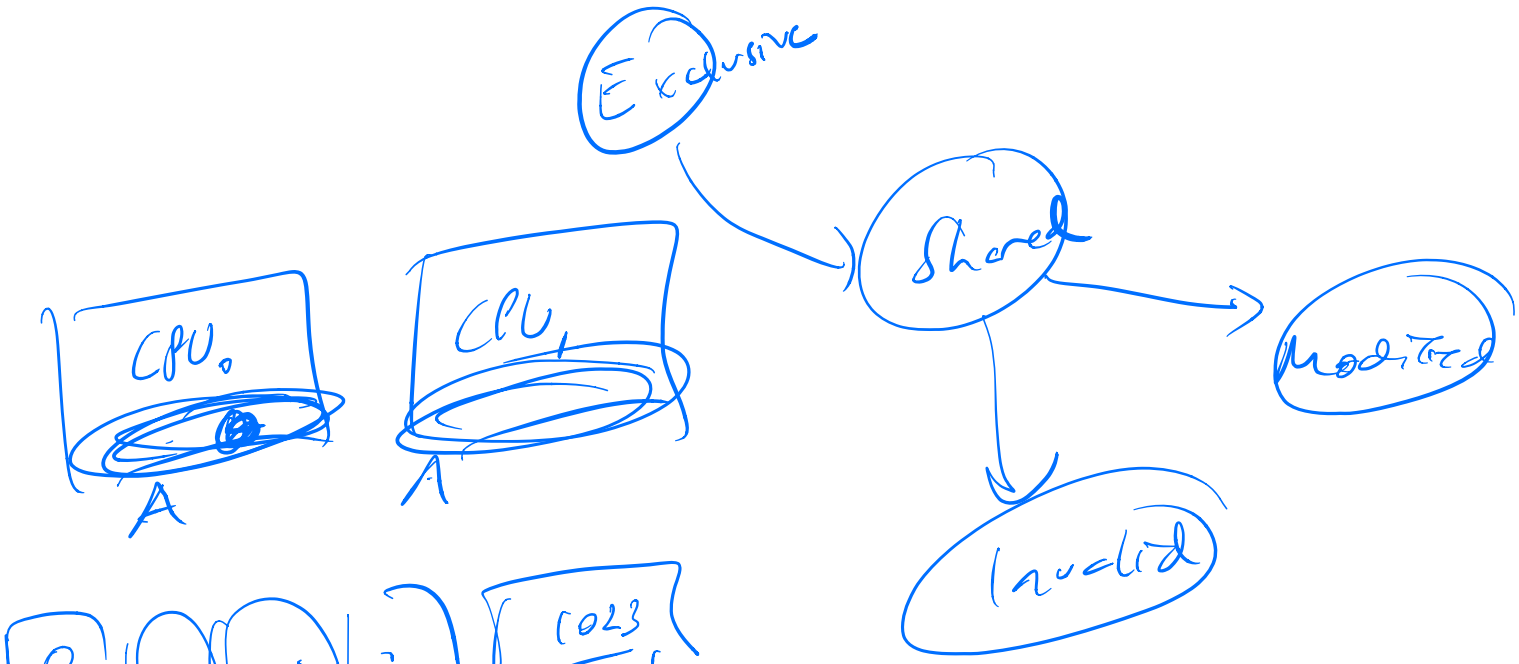
inclusive — copy in all levels
Intel

exclusive — copy in one level
AMD

"LRU" eviction

cache coherence protocol

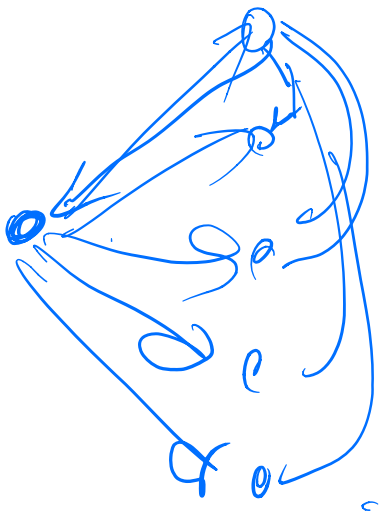
MESI



Scaling

scalability wrt N
⇒ execution time

$\sim$ polylog(N)

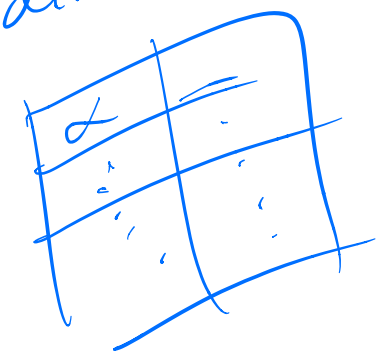


$N(N-1) \simeq N^2$

directory-based

scalable

$\sim$ hash table

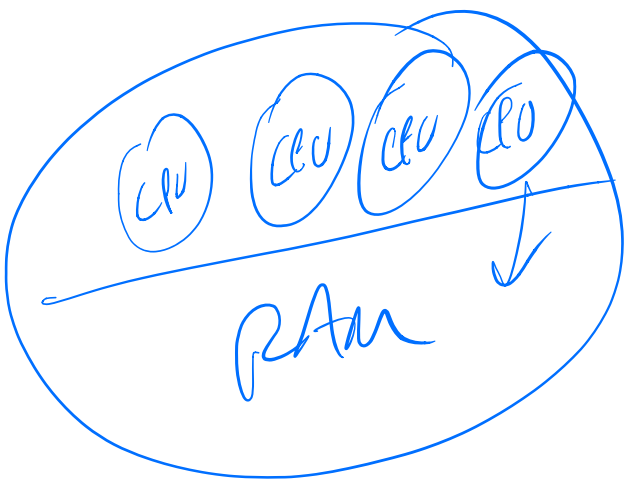


INCLUSIVE

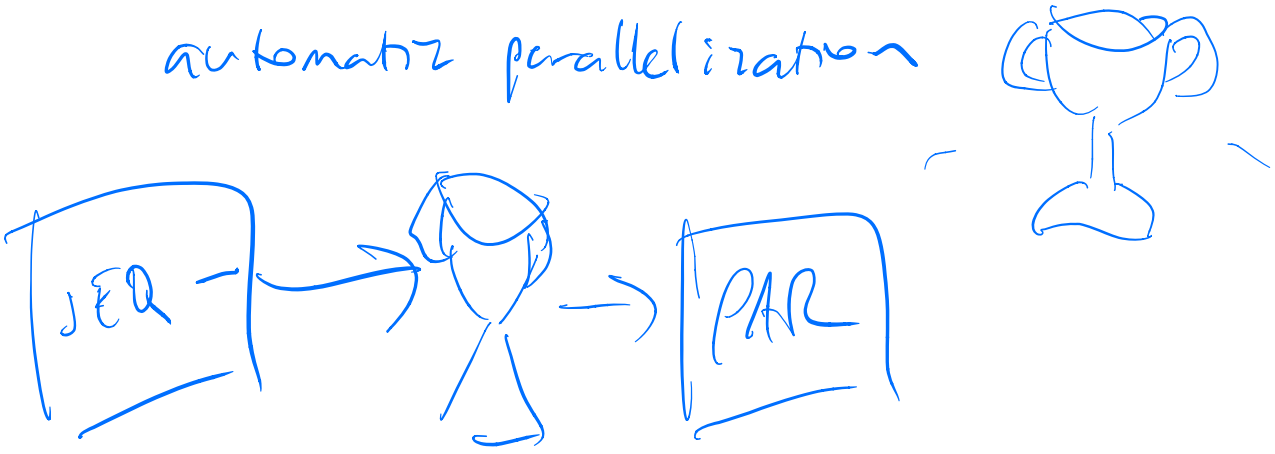
EXCLUSIVE

Cache details

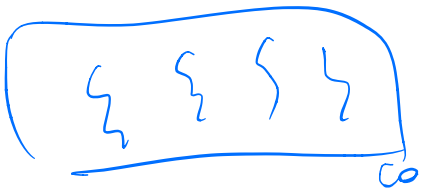
Amdahl's Law



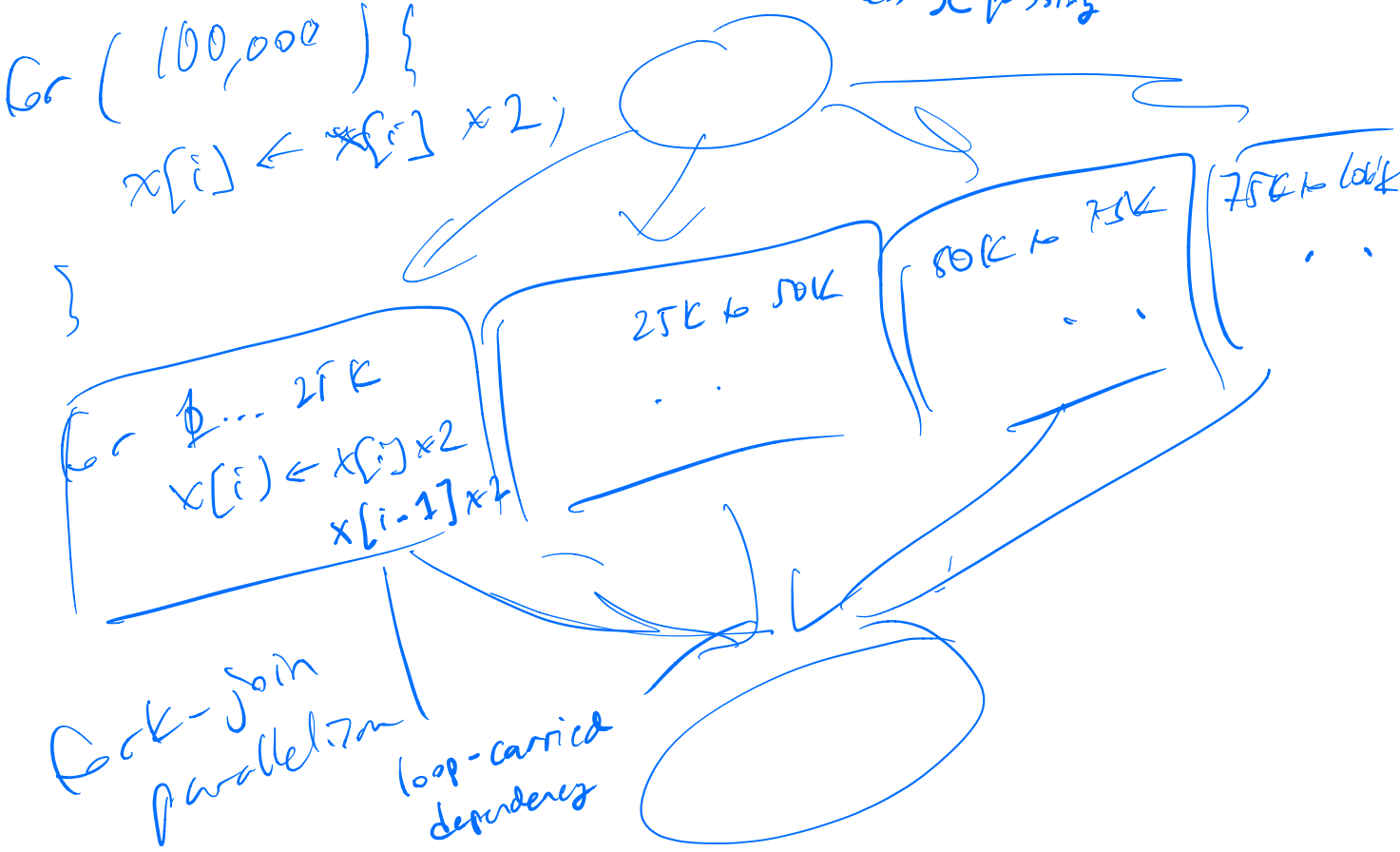
automatic parallelization

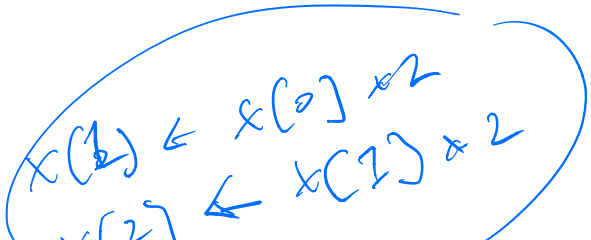


Multithreading

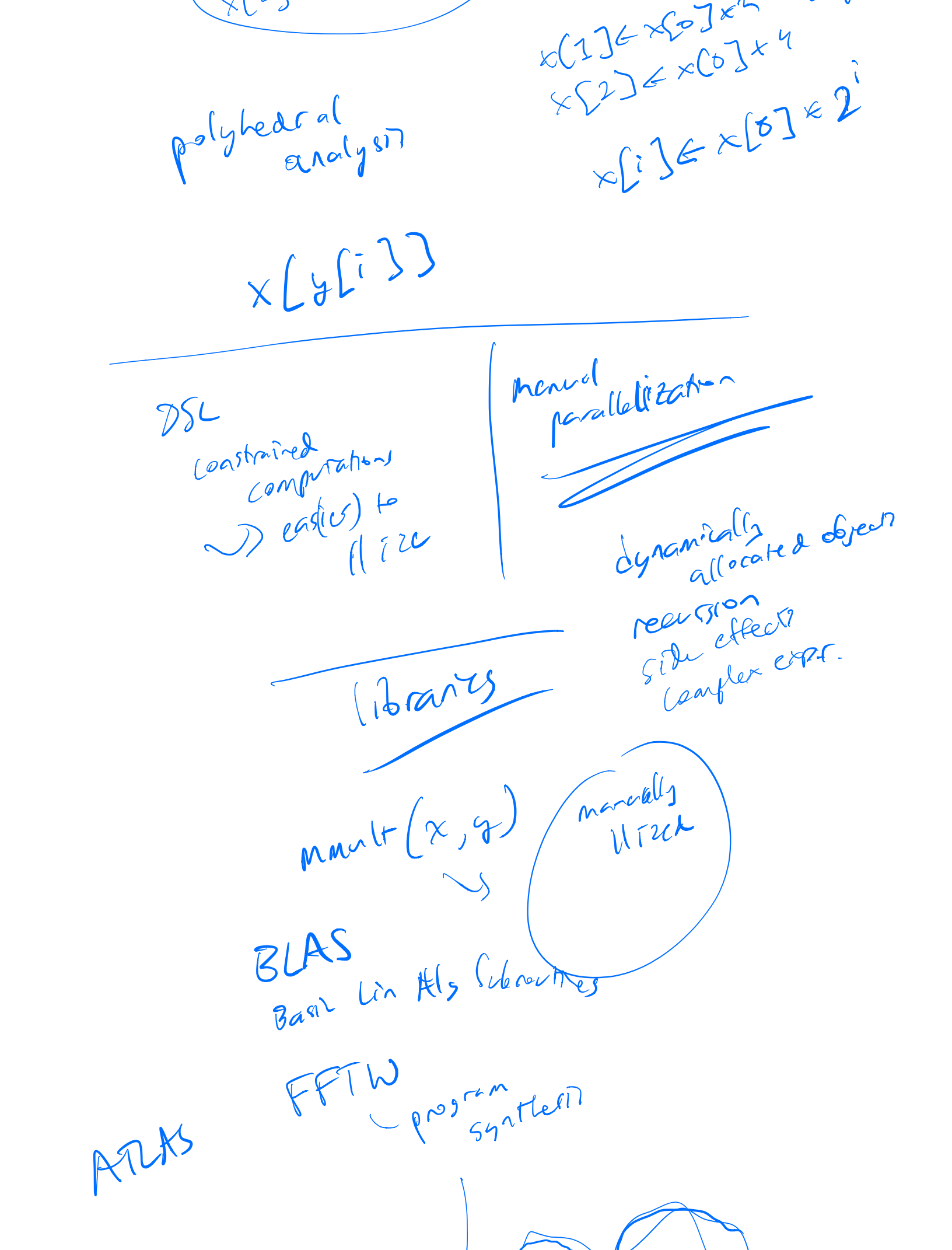


Multiprocessing





transform to eliminate loop carried dependencies


x[1] ← x[0] * c
x[2] ← x[0] * 4
x[i] ← x[0] * 2^i
polyhedral analysis
x[y[i]]
DSL
constrained computations
easier to parallelize
manual parallelization
dynamically allocated objects
recursion
side effects
complex expr.
libraries
mmult(x, y)
manually tuned
BLAS
Basic Lin Alg Subroutines
FFTW
program synthesis
ATLAS

$A = 3 \times C + D \times E$

correctness

threads
processes

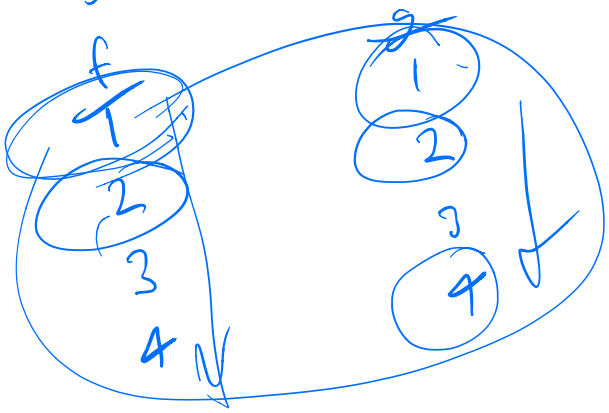
races

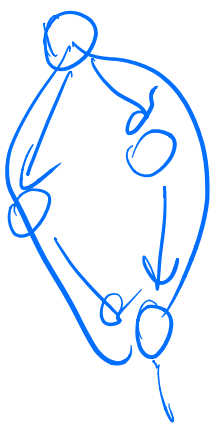
nondeterministic

performance

```
E() {
  x = 1;
  printf("%d", x);
}
```

```
G() {
  x = 2
}
```





f1 g1 s2 s3 g4 f2 f3 f4

$\sim \frac{2^4}{T^1}$ "1"

Synchronization
Contention (cache line level)
memory level
] SEQUENTIAL part

Thread t1 = new Thread(t, arg1);
Thread t2 = " " (t, arg2);

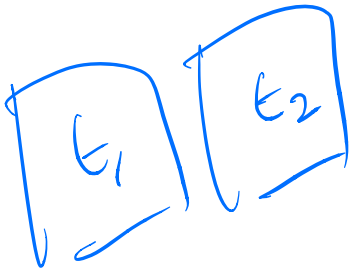
t1.wait()
t2.wait()

$T \ll P$ — insufficient # of threads
$T \gg P$ — too many threads
$T \overset{?}{=} P$

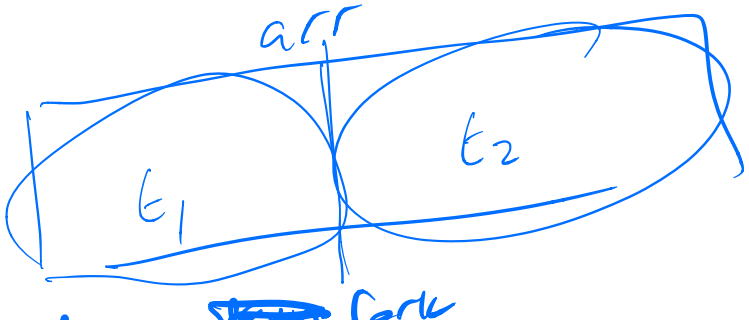
amortize cost

too fine-grained

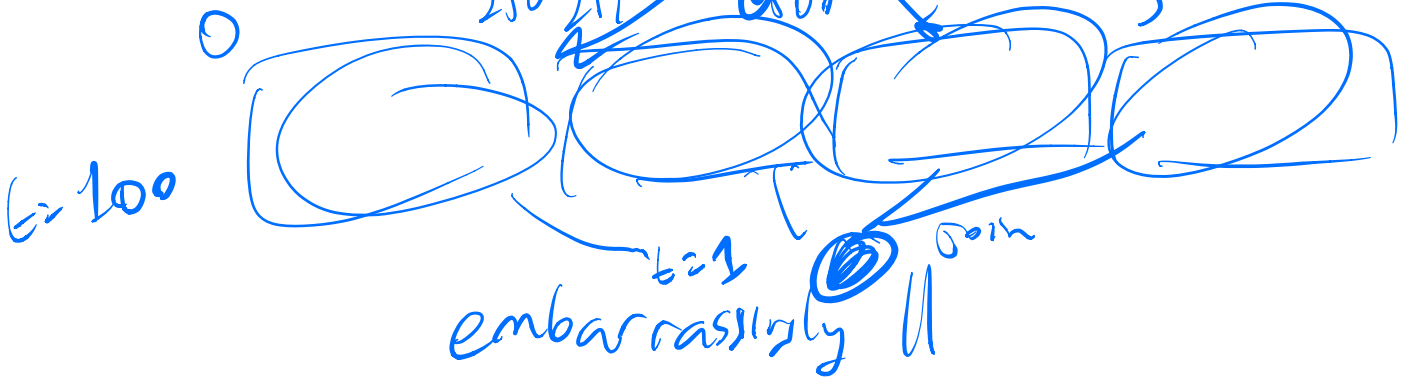
granularity



coarse-grained
$T \approx P$



$t = 1$



T_1 time with 1 processor (serial)

T_P time with P "

goal: $T_P \approx T_1/P$

T_∞ — "critical path"

$$T_P \approx \frac{T_1}{P} + O(T_\infty)$$

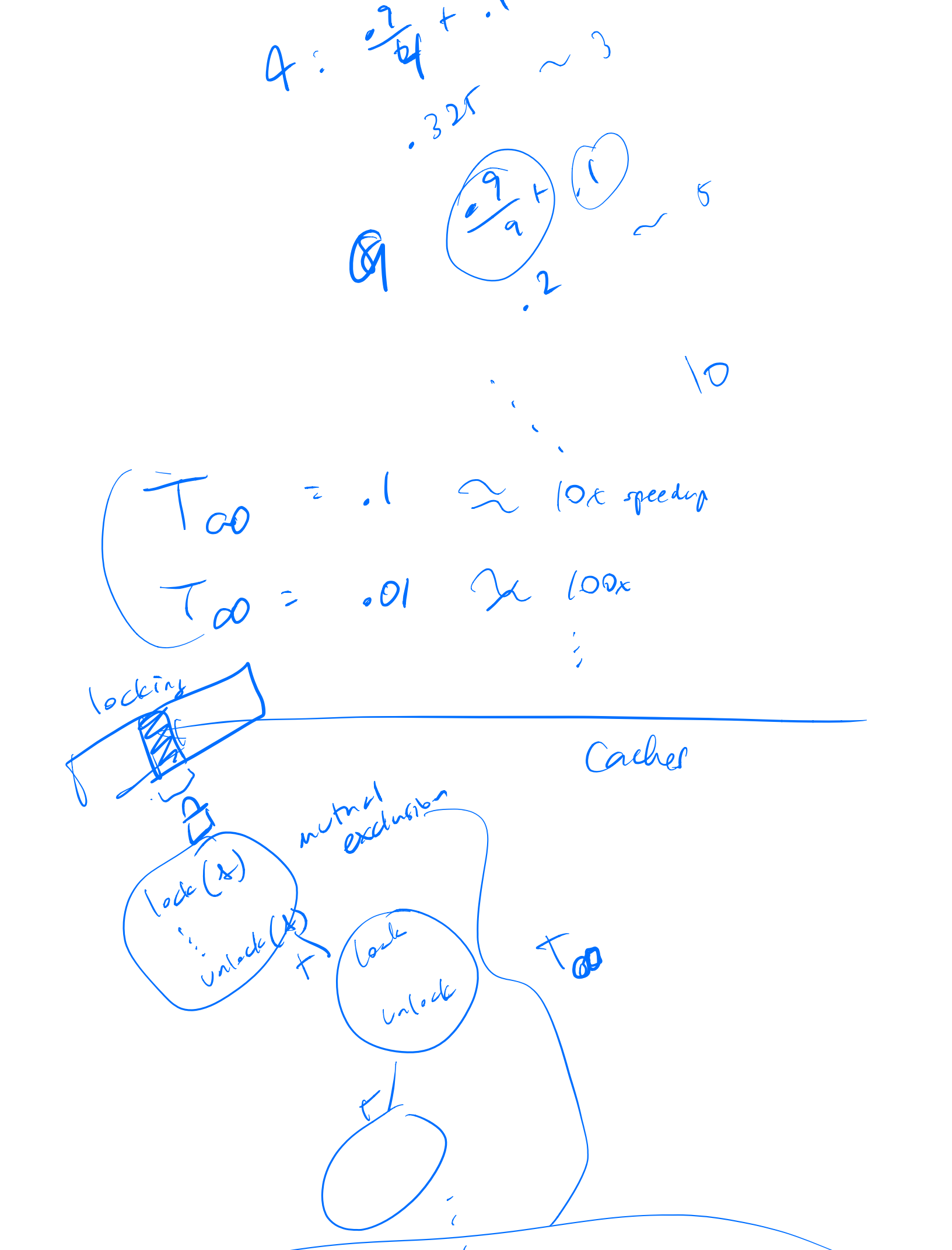
$\frac{T_S}{T_1}$ $\quad \frac{P}{P-S}$

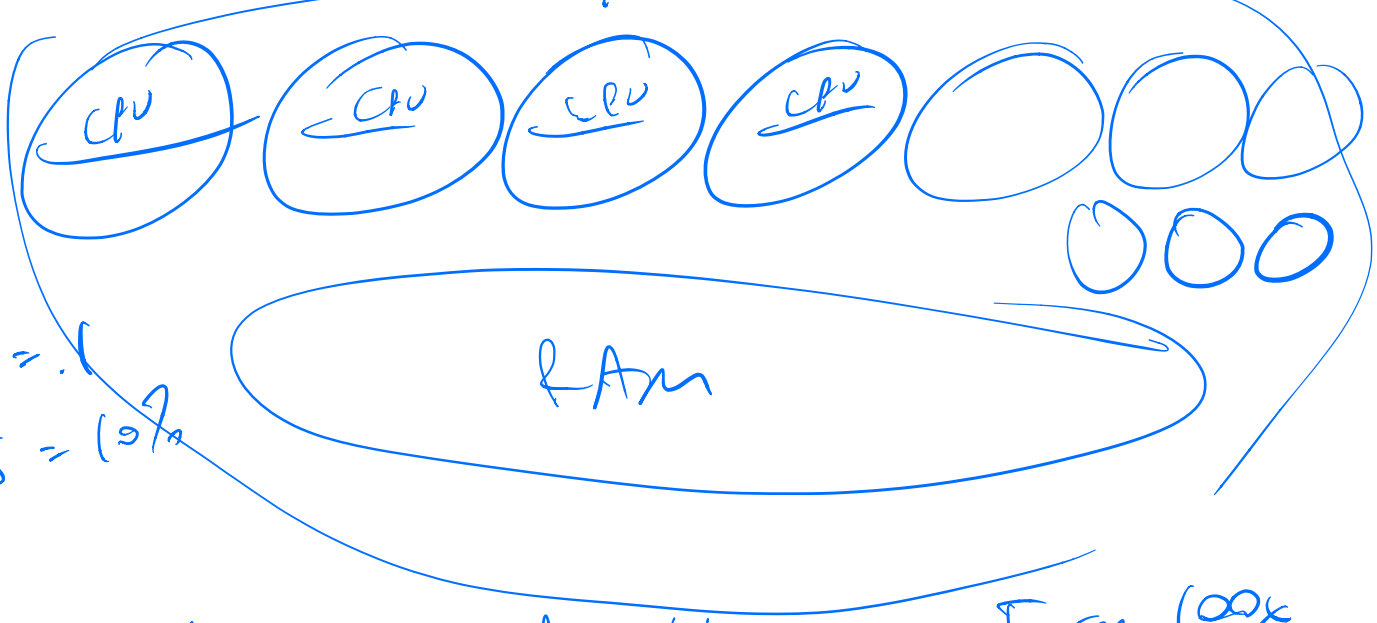
.9 .1

1

S_2: $\frac{.9}{2} + .1$

.55 ~ 2


$T_\infty = .1$ ≈ 10x speedup
$T_\infty = .01$ ≈ 100x
locking
Caches
mutual exclusion
lock(x)
unlock(x)
lock
unlock
T_∞



$t_0 = .1$
$S = 10\%$

Amdahl $\leq 10\times$ I see $100\times$

problem size

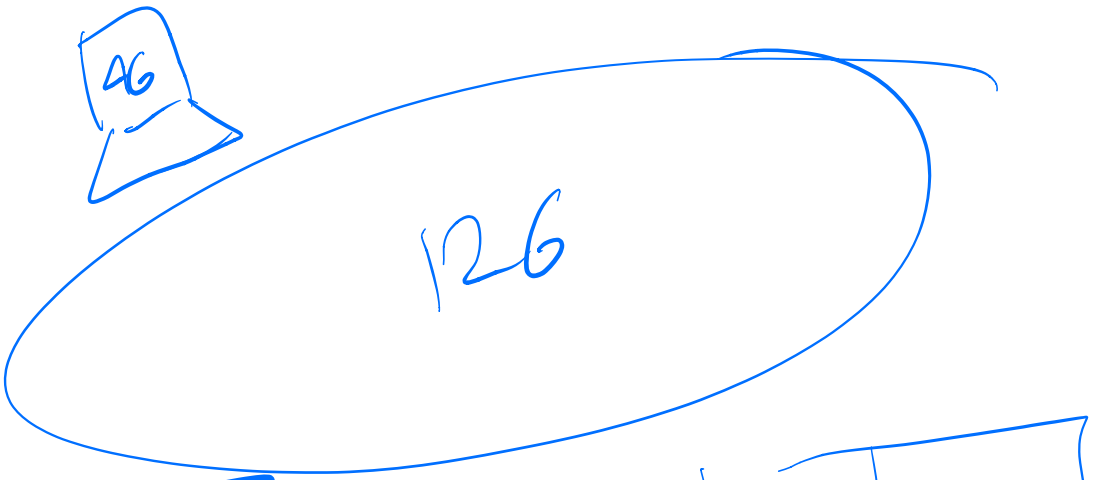
$\sim 10\times$

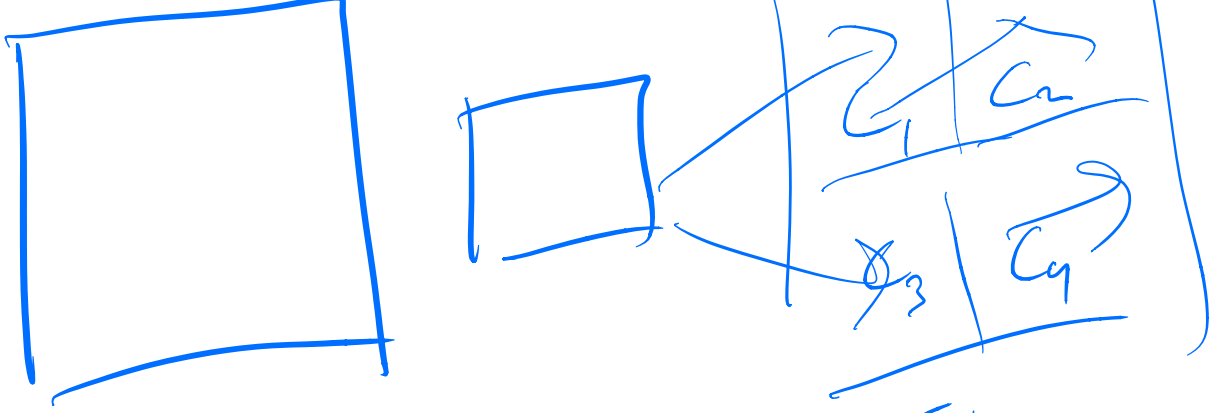
size of cache

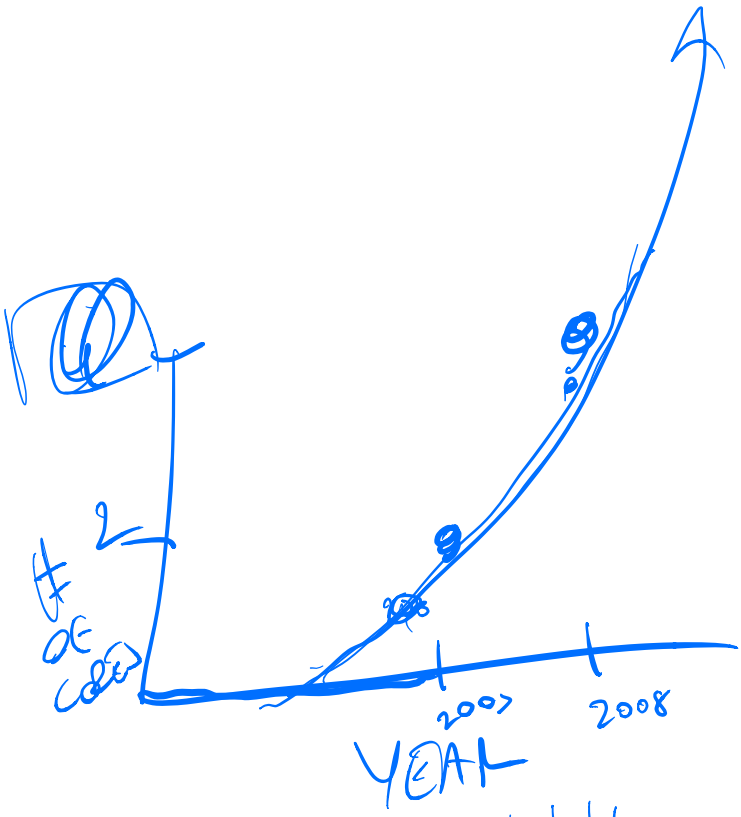
superlinear speedup

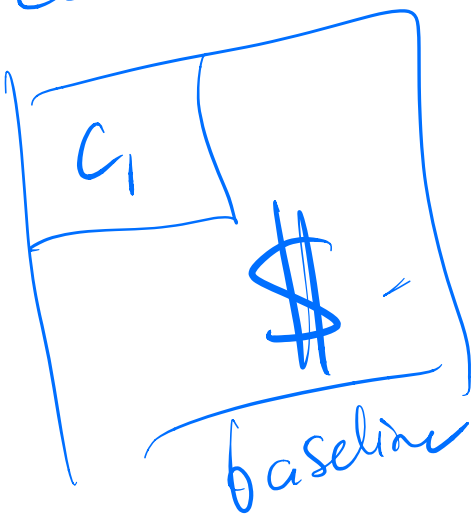
disk bound

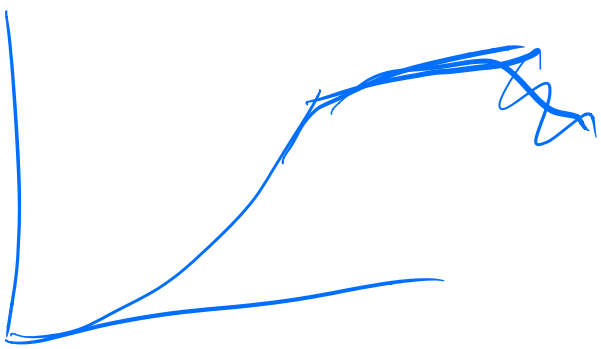
CPU bound

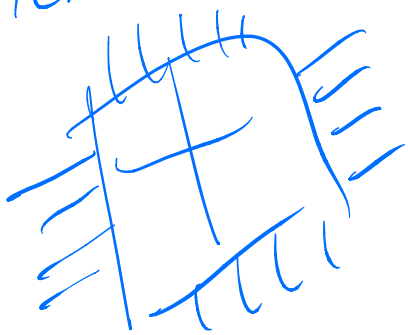


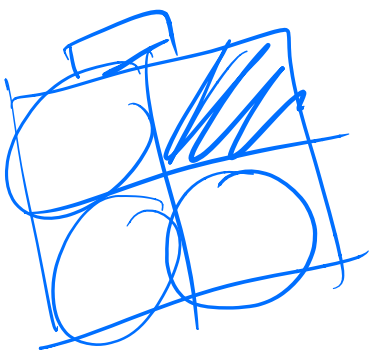












7 core
5 core
7 core

Caches

dataflow

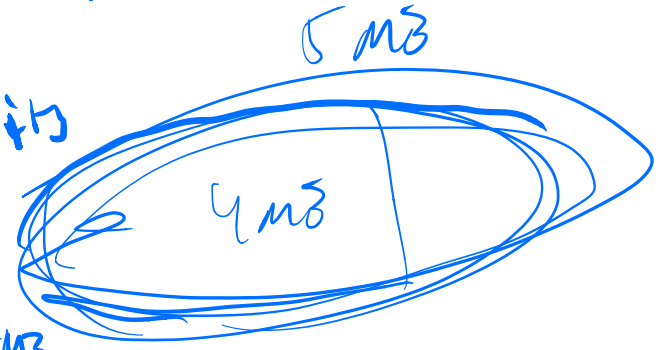
Cache misses

CAPACITY

COMPULSORY First access

CONFLICT miss

↳ not enough associativity (full)

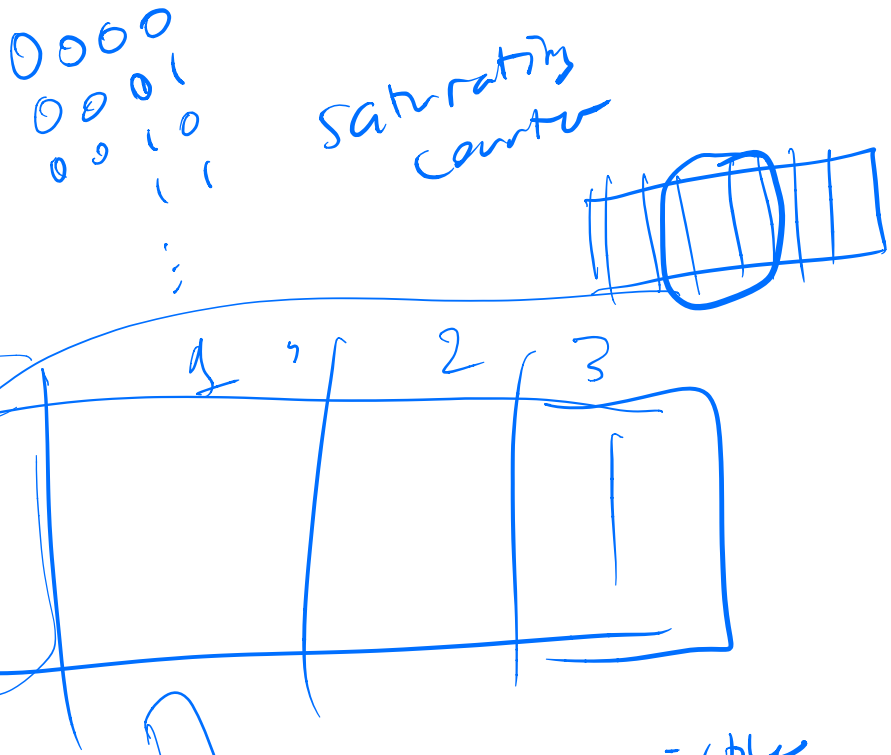


COHERENCE miss

time L3 4MB

LRU

MTF move to front

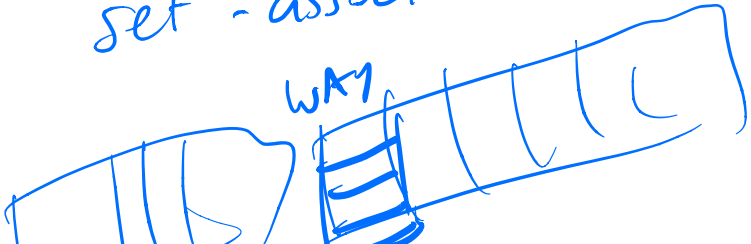


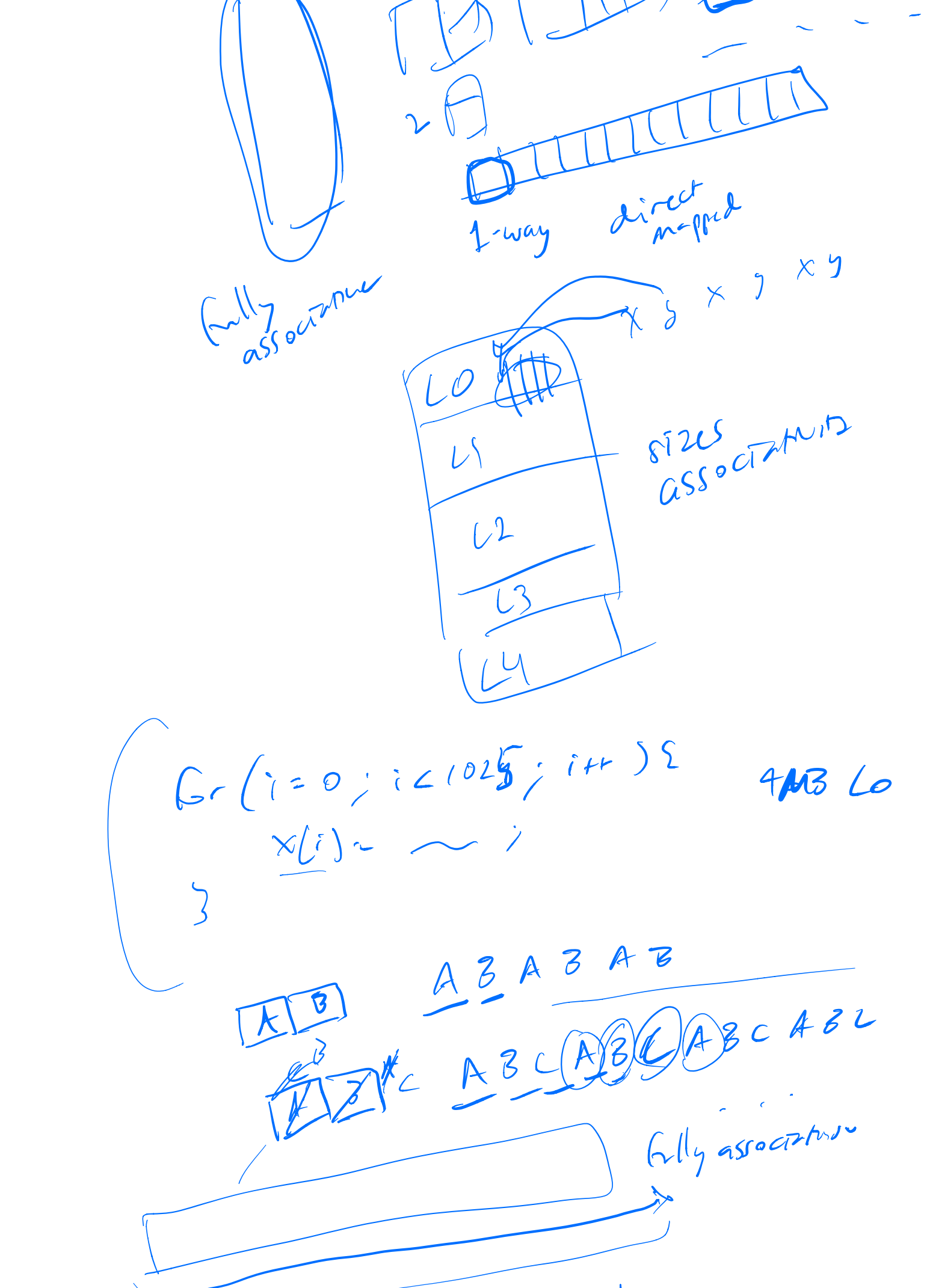
sets 8

set - associative

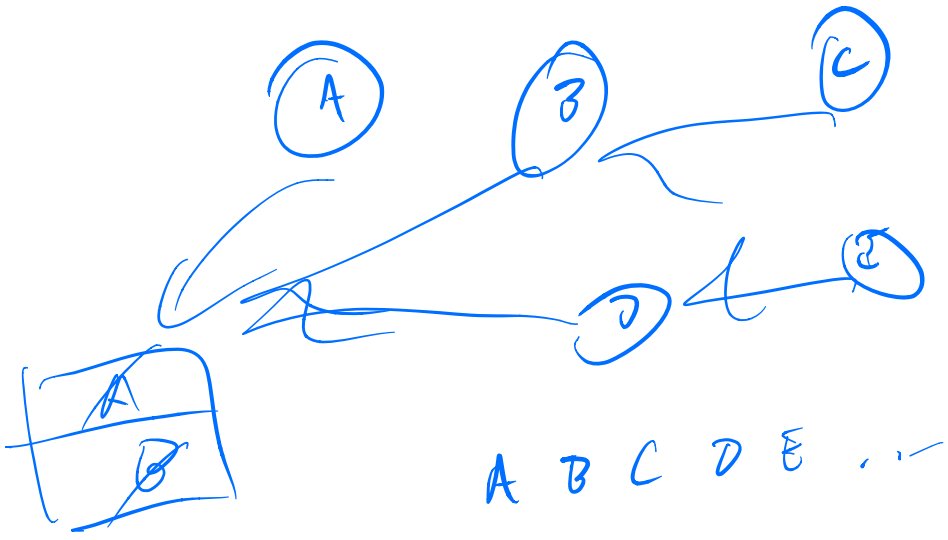
4

WAY



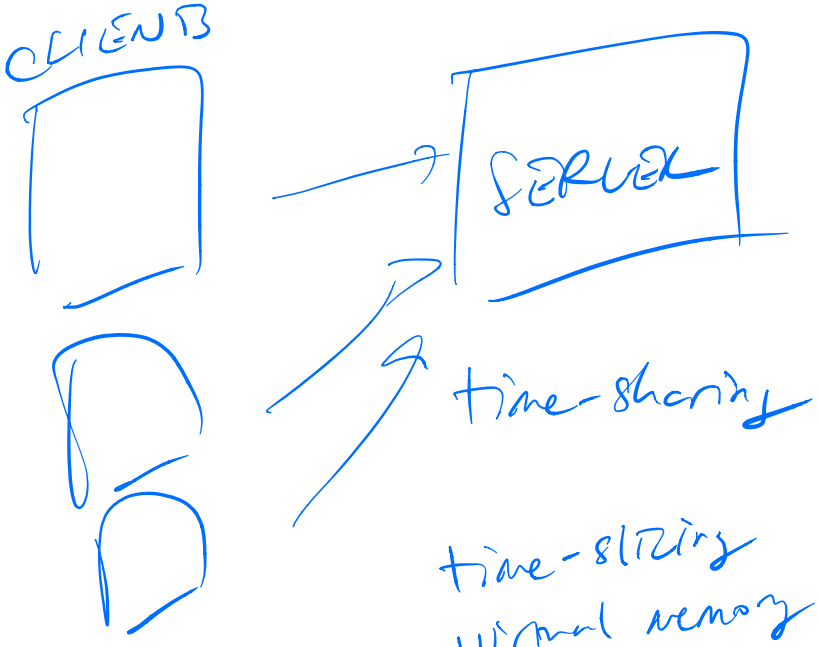

2
1-way direct mapped
fully associative
L0
L1
L2
L3
L4
sizes associativity
for (i=0; i<1025; i++) {
x[i] = ~ i;
}
4MB L0
A B
A B A B A B
A B C A B C A B C A B C
fully associative

pathological
worst-case

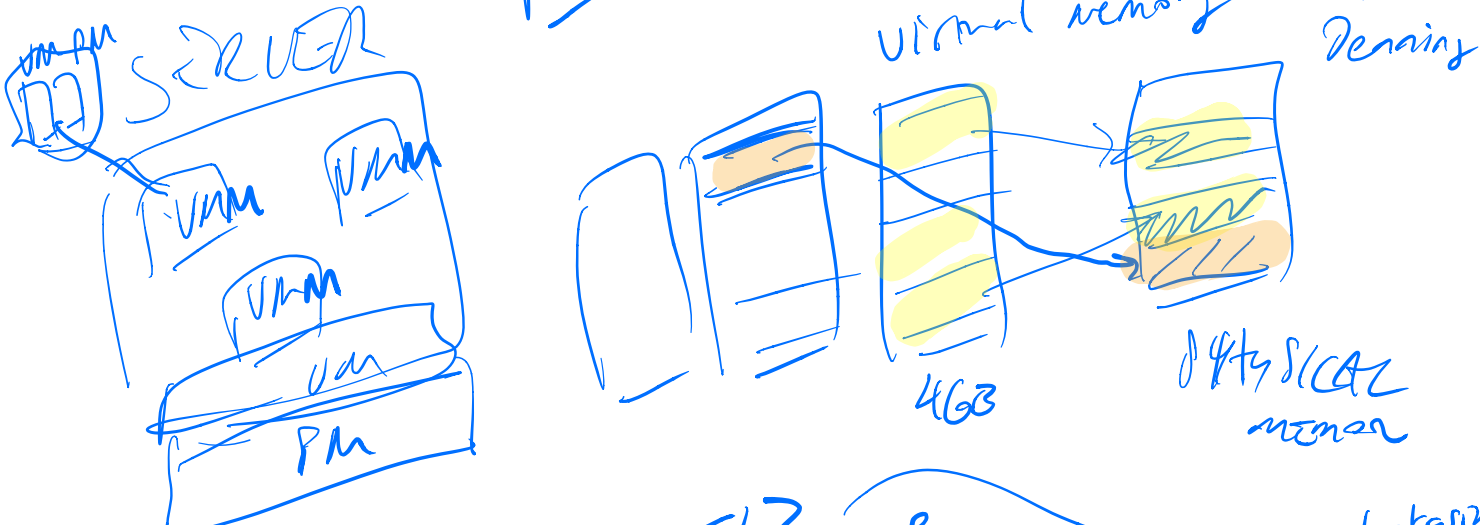


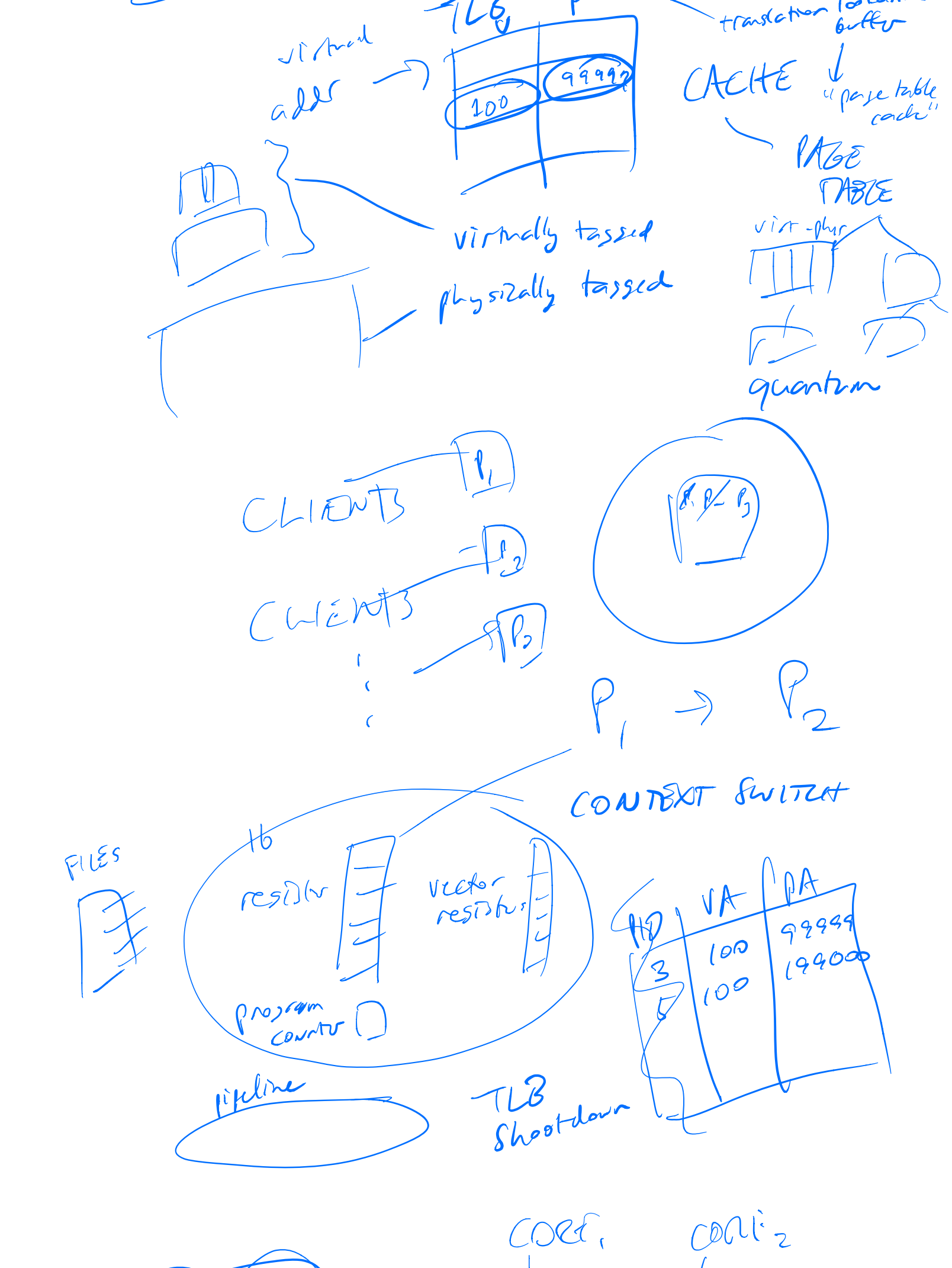
interlude

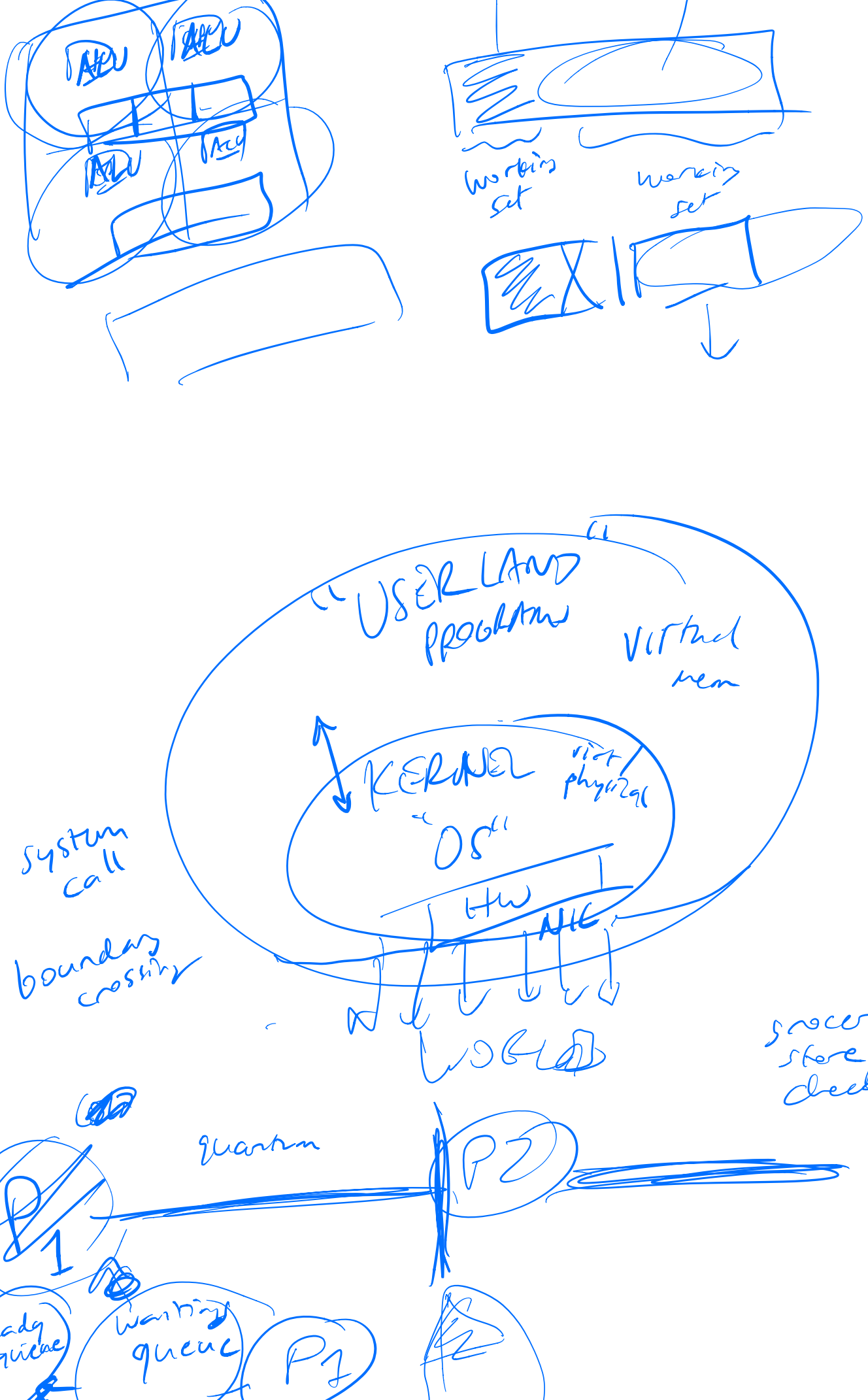
software architecture



time-slicing
virtual memory ~1968
Denning




TLB
virtual addr
100
99999
translation lookaside buffer
CACHE
"page table cache"
PAGE TABLE
virt-phys
virtually tagged
physically tagged
quantum
CLIENTS
CLIENTS
P_1
P_2
P_3
$P_1 \rightarrow P_2$
CONTEXT SWITCH
FILES
regs
vector regs
program counter
pipeline
TLB Shootdown
VA
PA
HD
3
100
100
99999
199000
CORE$_1$
CORE$_2$

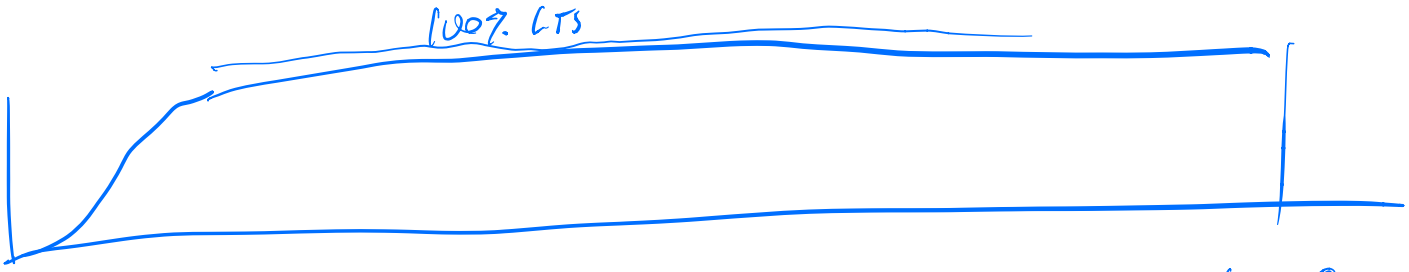

ALU
ALU
ALU
ALU
working set
working set
"USERLAND PROGRAMS"
virtual mem
KERNEL
"OS"
virt
physical
HW
MMU
system call
boundary crossing
I/O
process
store
checkout
quantum
P1
P2
ready queue
waiting queue
P3
P2

better responsiveness — lower latency

quantum — small

higher throughput "faster"

quantum — large

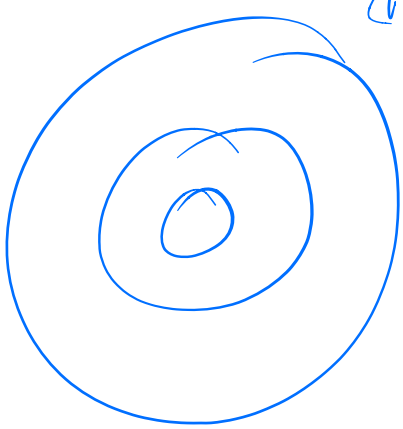


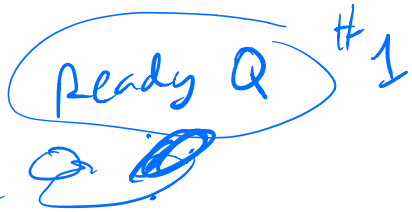
long q

(PROTECTION) RINGS

Multics

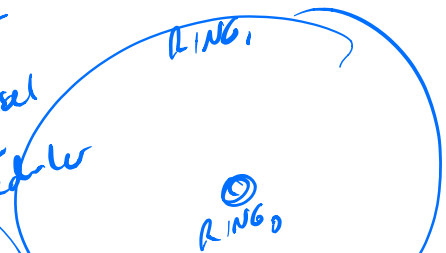
Bob Graham





priority based scheduling

proportional share scheduler



eventually runs

real time scheduler

preemption → PROGRESS

deadlines

Carl Waldspurger

STARVATION
— absence of progress

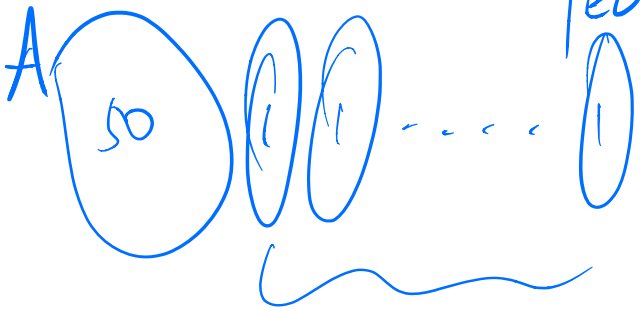
lottery scheduling

P_1 10ms

P_2 30ms

EDF earliest deadline first

greedy & optimal

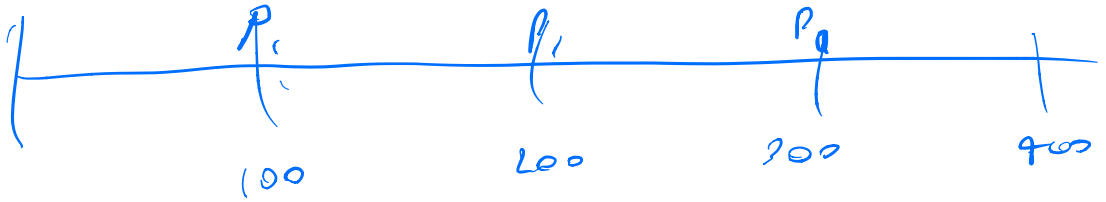


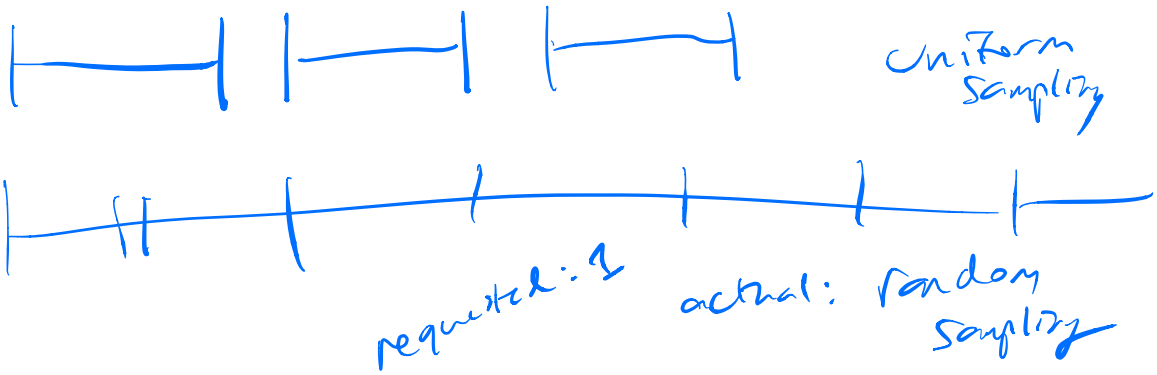
50

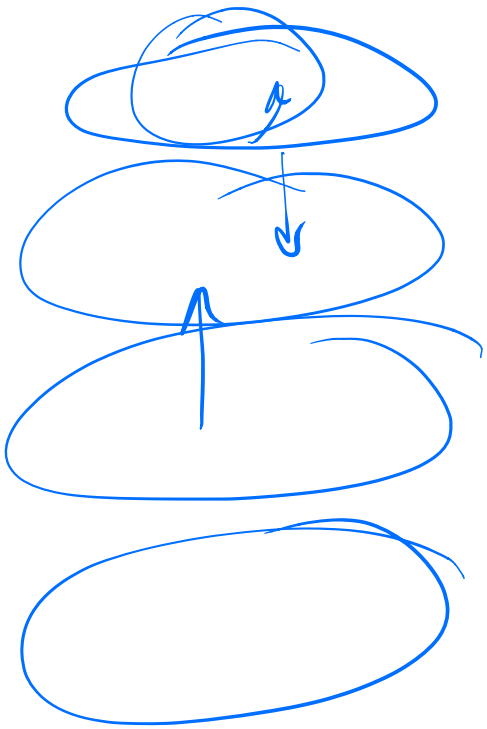
$Pr[A \text{ wins lottery}] \geq .5$

$A_i \in S$ $\quad corr(A_i, A_j) = 0$

$A_0 \ldots A_{n-1} \Rightarrow A_n$







1

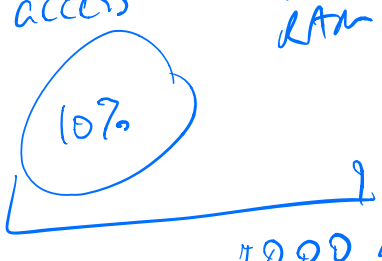
priority

100

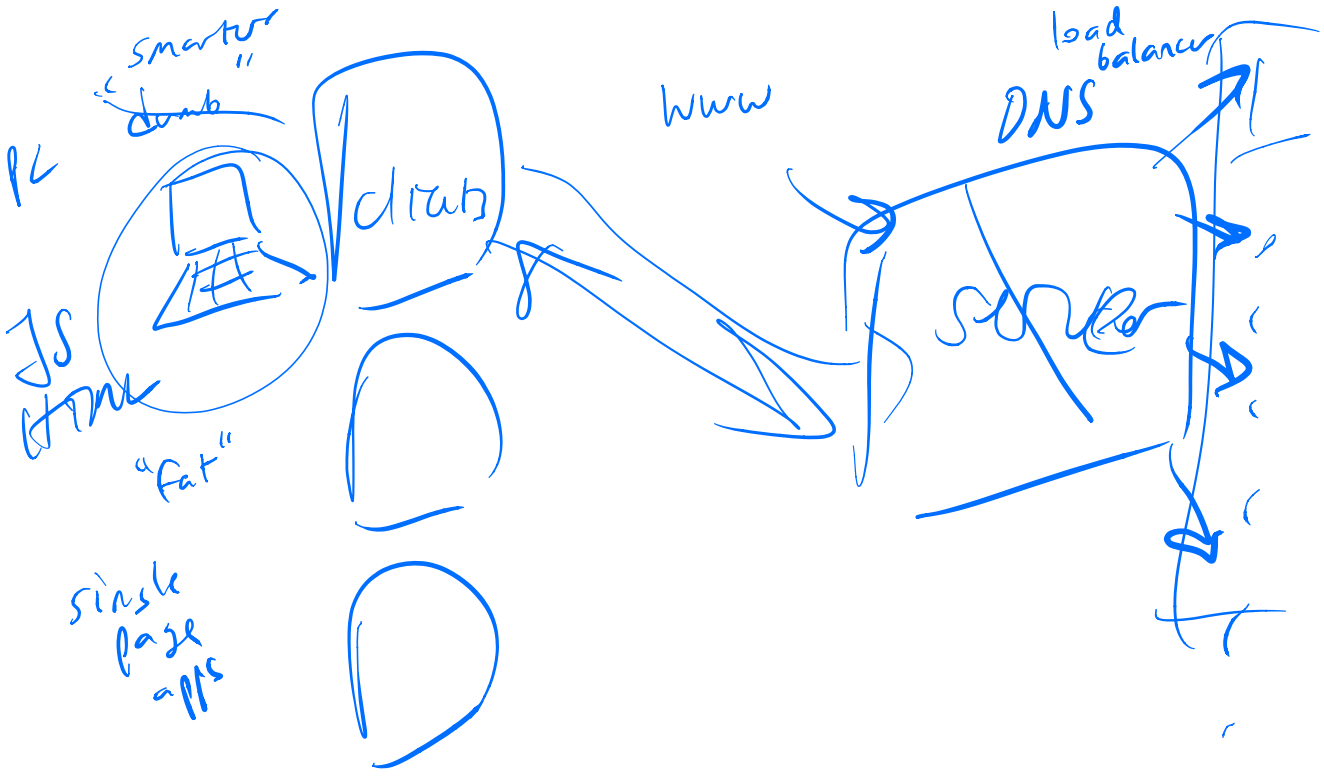
CPU utilization : 100%

QoS guarantee

quality of service



$Pr\left[\begin{array}{c}10\% \text{ service} \\ \text{every } 1000\text{ms}\end{array}\right] \geq$!



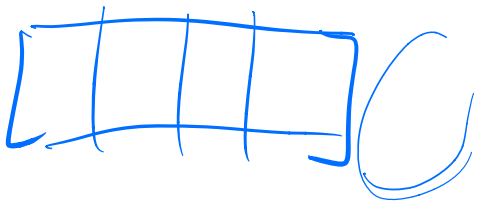
memory leak

Bleak

---

replacement mem allocator for Rust (in Rust)

"safe" language



X bounds checking

mem mgmt "safe"

no dangling pointer errors

Rust → no GC

explicit
malloc
fast, clear
out of control

OWNERSHIP TYPES

GC
safe
"fast"
mem leak

smart pointers
C++

"mut"

unsafe

traits

malloc
free
getsize

directly
as replacements

My malloc
my free
my getsize

memory

bump pointer

p1 8 p2 16 p3 24

THREAD SAFE

free(p1)

linked list
"free list"

16
32
64
:
32

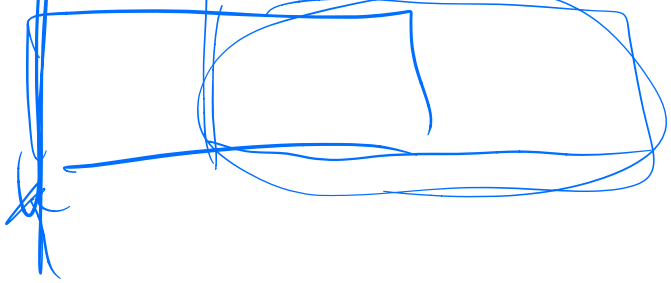
power of
two size
classes

17

$2^i + 1 \dots 2^{i+1}$

2^{i+1}

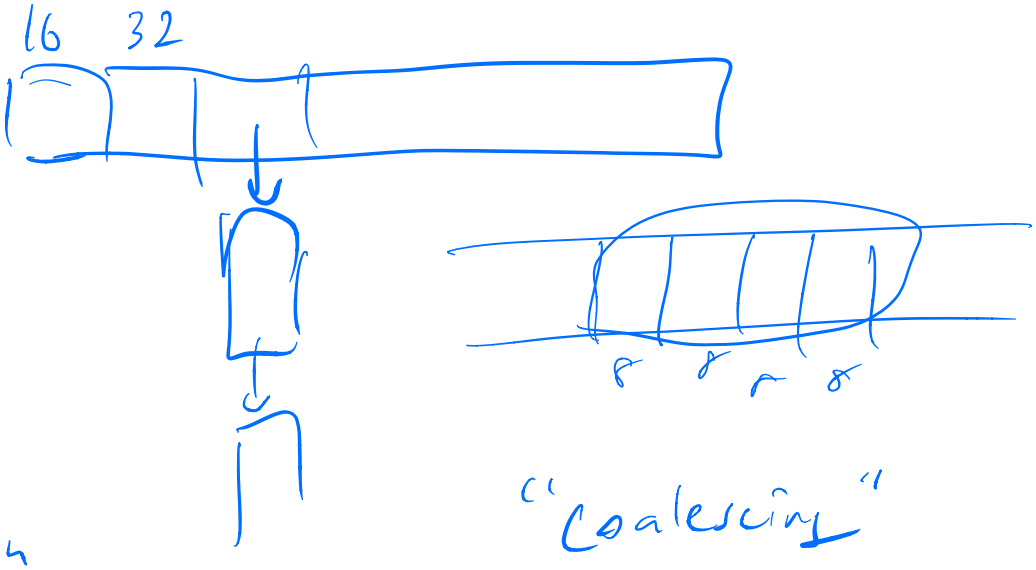
OBJ HEADER



all objects must be "aligned"

SEGREGATED FITS (BEST-FIRST) ALLOCATOR

4-byte 0 8
16-byte 0



Paul Wilson 92/94

surveys of uniprocessor GC mem allocator

adjacent small freed objects → one big freed object

"coalescing"

Robson 1972

$\left(\log \frac{M}{n}\right)$ biggest / smallest

M × n

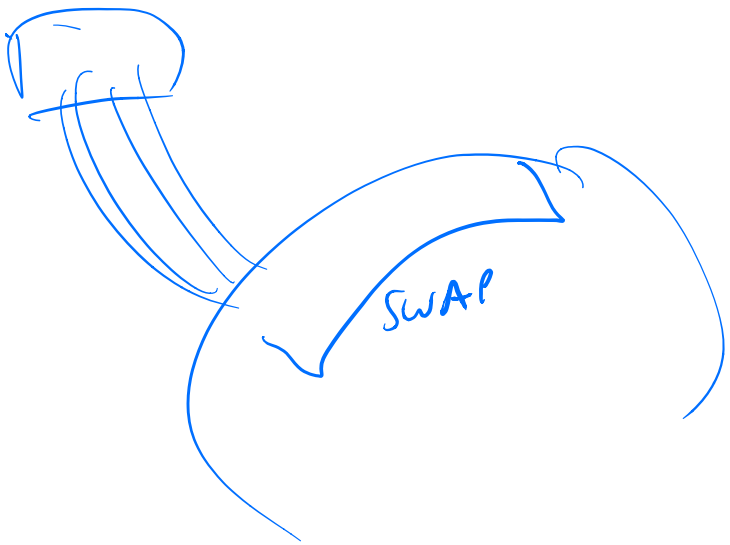
memory

power-of-two size classes

~~int[4]~~

arrays (stacks)

64 bit systems



OS — hey I exist

Memory mapped "mmap"

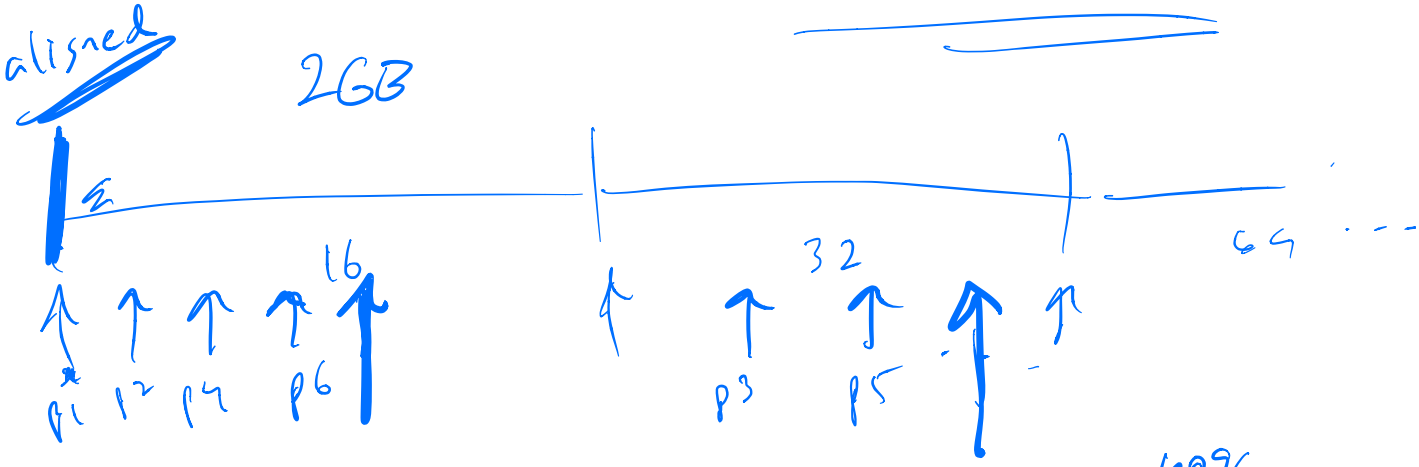
Madvise (DONTNEED)

Readable writeable anonymous private

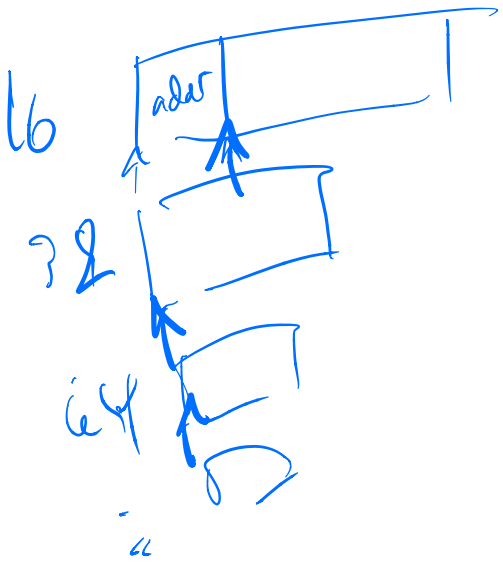
MAP_NORESERVE

aligned

2GB



4096 byte



malloc: if stk ! empty
pop stk & ret ptr
else
bump memory ptr
& return

free: push ptr onto stack

inUse
maxAllocated

"CHeap"

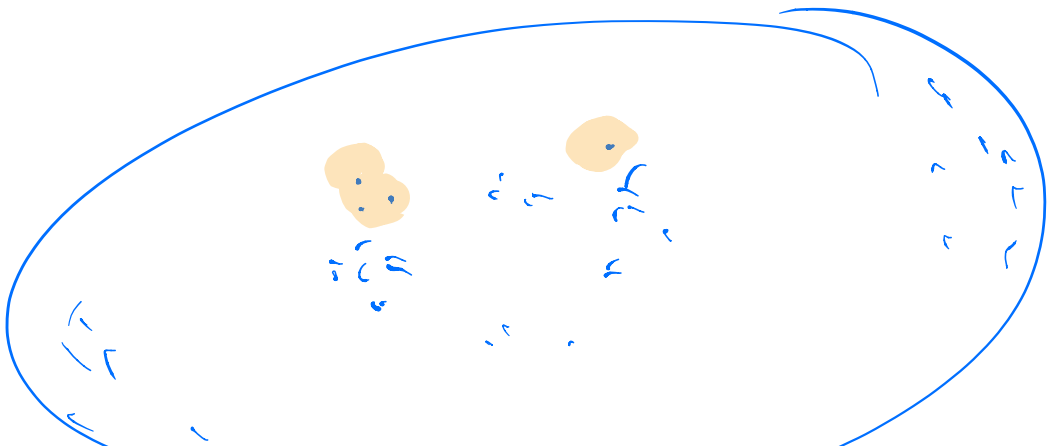
zero on-demand page

Dave Patterson
Hennessy

RISC
RAID

reduced-instruction set computing

every new instruction for Ute



CISC

Conventional instruction set computing

"BAD for compiler writers"

RISC-V

Krste Asanovic

RISC

2000s

Intel Xeon

ARM

power

energy consumption

very few addressing modes

same size

MOV

ADD

fetch

decode

execute

8088

8086

80x86 ...

SSE

SSE2

SSE3

crypto

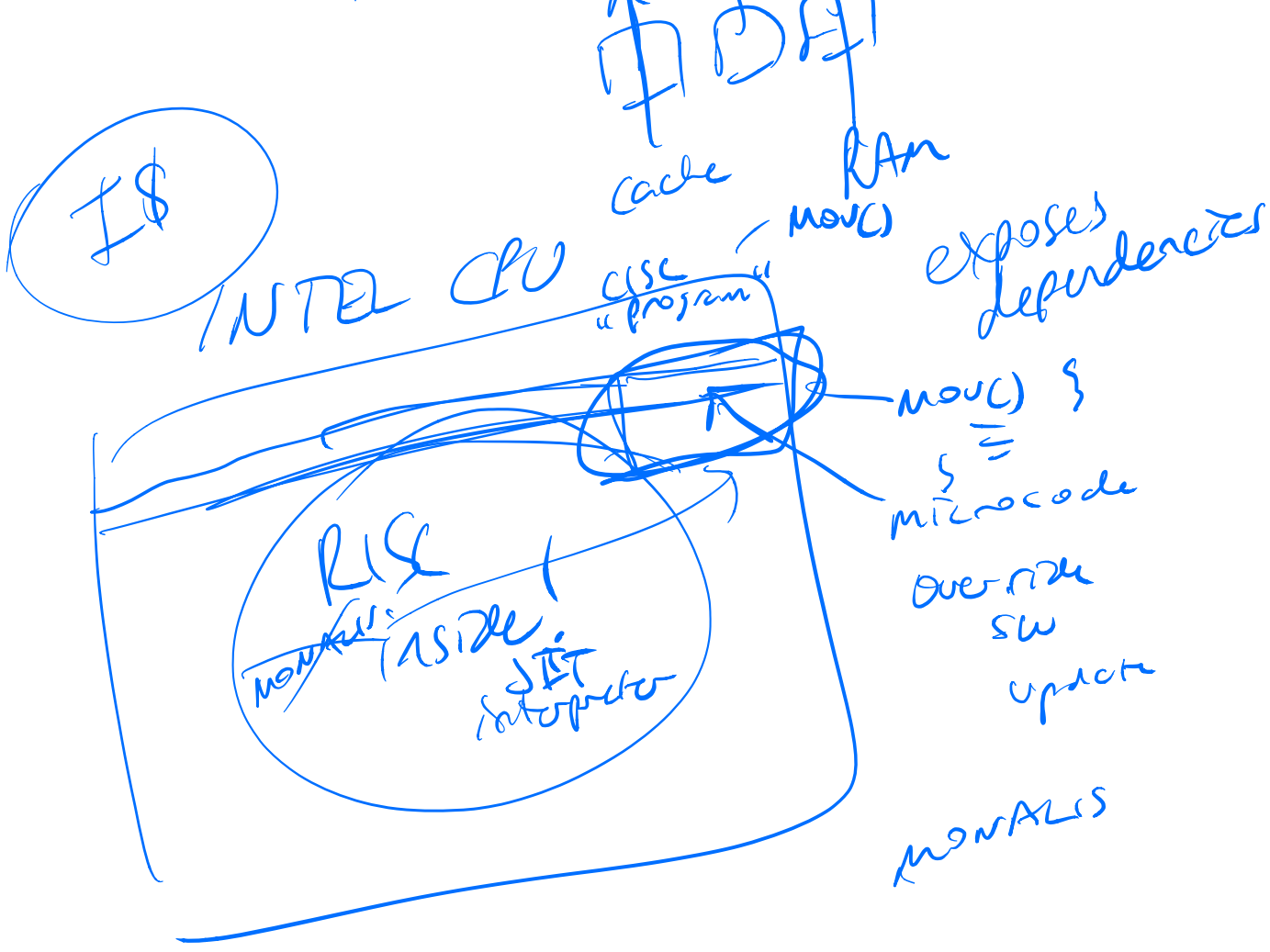
MMX

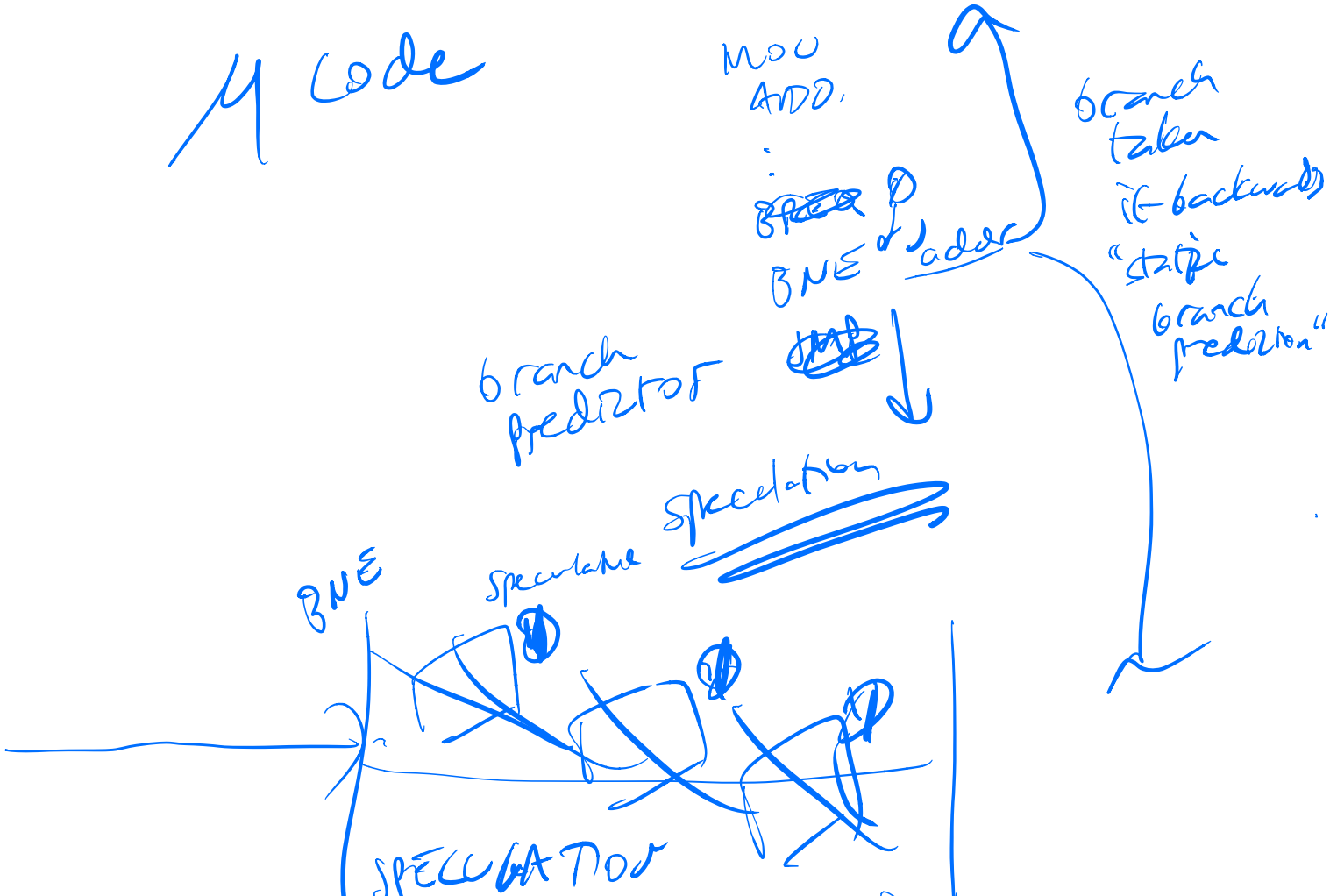
TSX

I$

ROP

pipeline


I$
cache
RAM
MOV()
INTEL CPU
CISC "program"
exposes dependencies
MOV() {
}
microcode
RISC
MONARCH
INSIDE
JIT
interpreter
overrides
SW
update
MONARCH


μ code
MOV
ADD.
.
.
BNE
loader
branch taken (backwards)
"static branch prediction"
branch predictor
speculation
BNE
speculate
SPECULATOR


speculative
state
YES!
NO