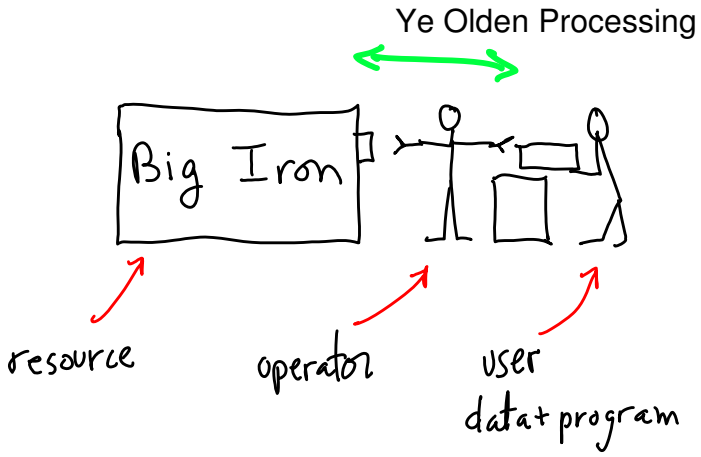
Parallel Processing

1. Definitions
2. Goals
3. Types



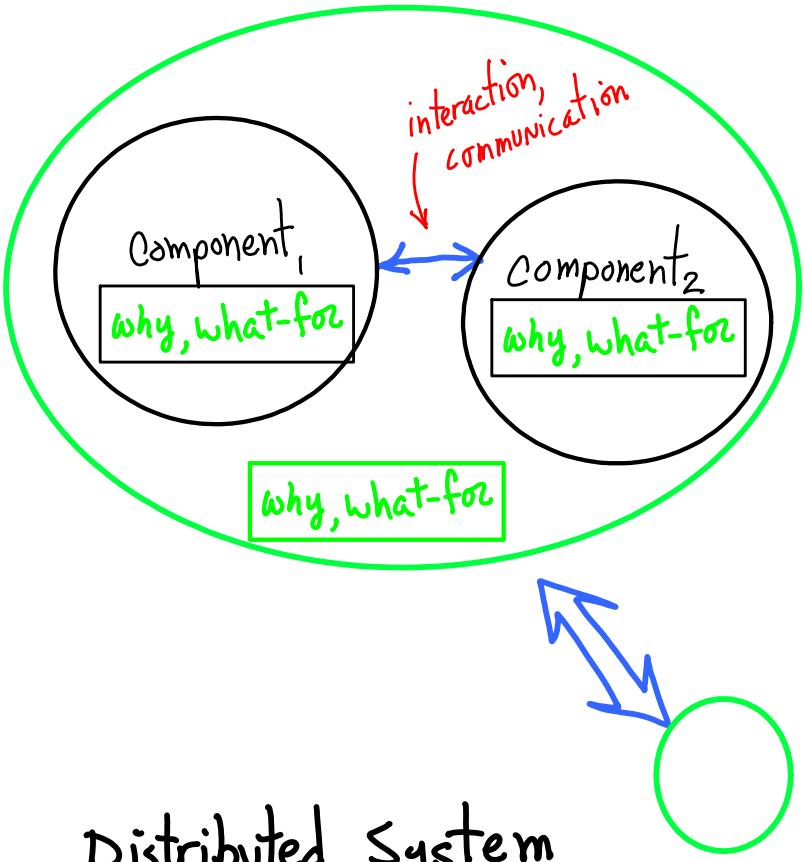
a system, generally

component = internally highly connected, not easily separable, functionally distinct

communication = relatively sparse connections between components

Hierarchy of
- Structure: grouping of components
- Purpose: why, what-for

Surfaces are logical and/or time-like and/or space-like



Boundaries are drawn arbitrarily, by us, for some specific purpose.

Distributed System

A system w/
- s/ow communication between components (distance [logical / physical])
- Less common dependency, more autonomy (no shared, common dependency on a single component) ★
- many components ≈ peers

- United by common/shared goals
- separated by local clocks
  local memory

☆ [more constant* vs less constant] *ignored

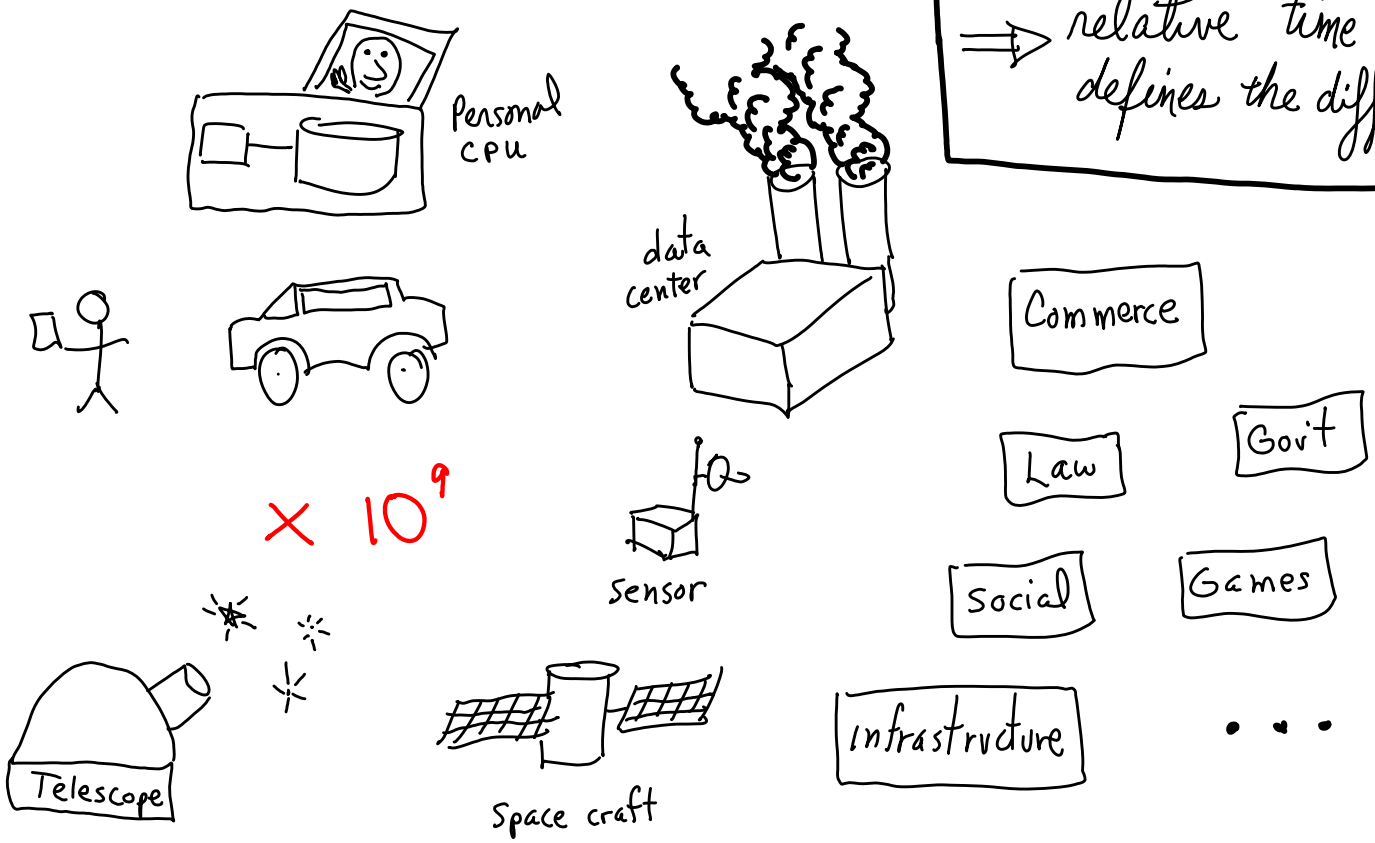
Common, but constant: Earth *

Common, but not constant: power switch

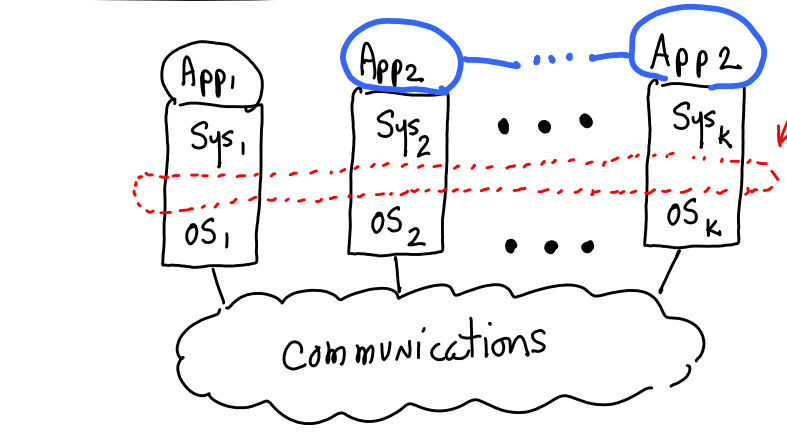
⇒ relative time scales defines the difference

Systems Today are heterogeneous

Internally and externally



× 10^9



middle ware

logical interface : hierarchy

One version of a (slightly) larger view of our reality.

Parallelism is everywhere!

Goals - Accessible resources, sharing, efficiency, reliability

- collaboration on shared enterprise - But, with control + security

Requires (?) - Scalability, modularity, incremental expansion, heterogeneity

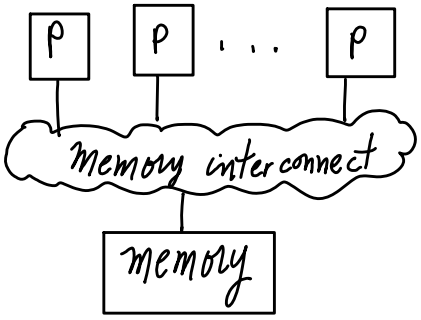
- Transparency (not visible)
  - access: hide data representation differences [types, file names, methods]
  - Location: hide latencies, specific details of location/access path
  - migration: hide dynamic reconfiguration
  - relocation
  - replication: hide caching, manage consistency
  - concurrency: hide other's usage
  - failure: hide intermittent response

Parallel Systems

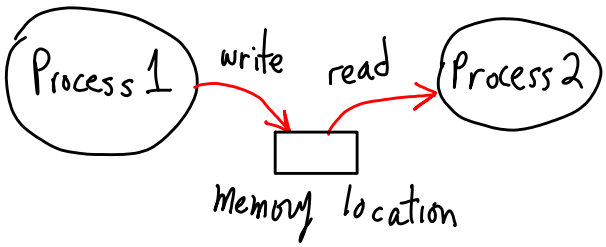
common address space

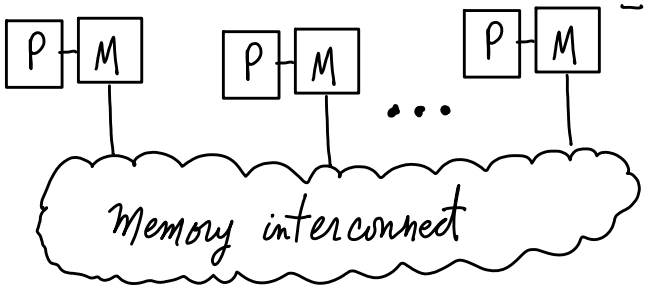
P = Processor | cache | cache | I/O

- multi-processor, shared memory (SMP)
  shared interconnect
  - local clocks / global clock



- Uniform access to memory
  UMA





- non-UMA:
  NUMA

coordination through memory
(can implement messages on top)

Classification

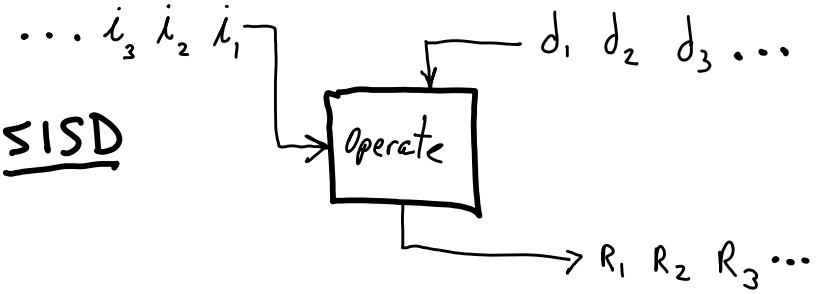
Hardware = serial Program = sequential	HW = serial SW = concurrent	HW = parallel SW = sequential	HW = parallel SW = concurrent
e.g. single core Matrix mult	e.g. single core OS multi-processing	e.g. multi-core Matrix mult	e.g. multi-core OS multi-processing

Taxonomy

Single Instruction, Single Data **SISD**

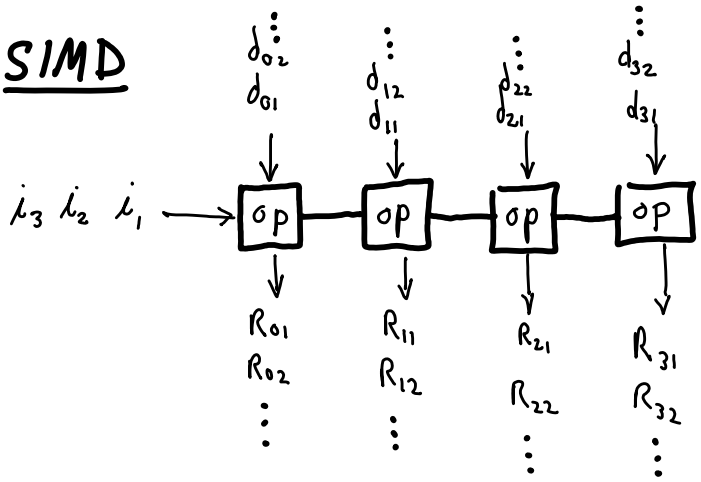
STREAM STREAM

(serial)



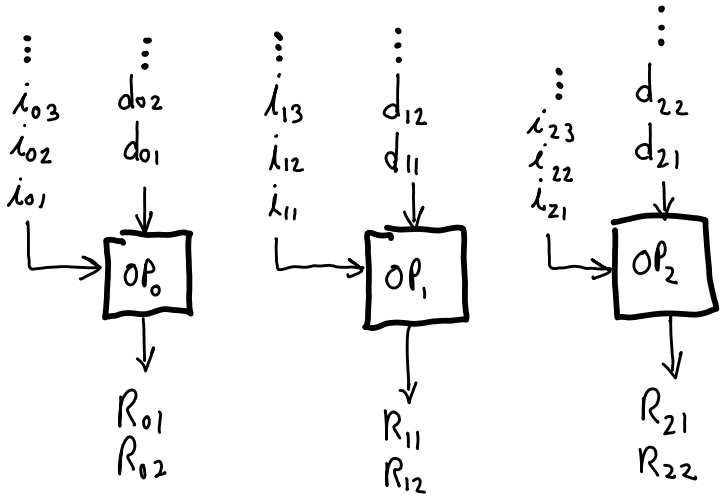
Single Instruction, Multiple Data **SIMD**

(data-level parallelism)

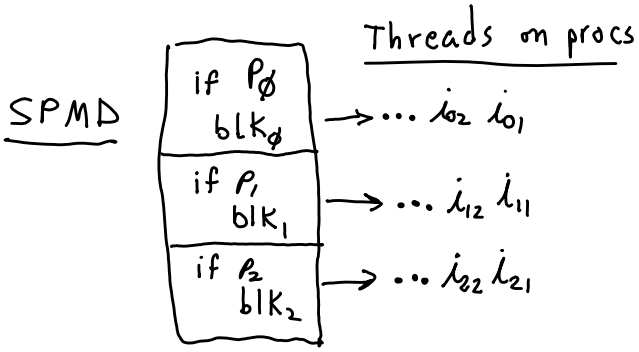


Multi-Instruction, Multi-Data **MIMD**

(instruction and data parallel)



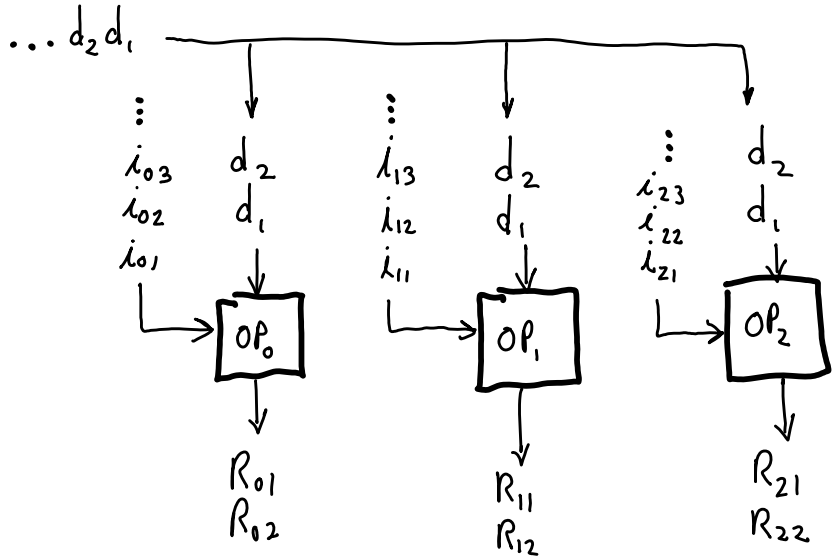
SPMD



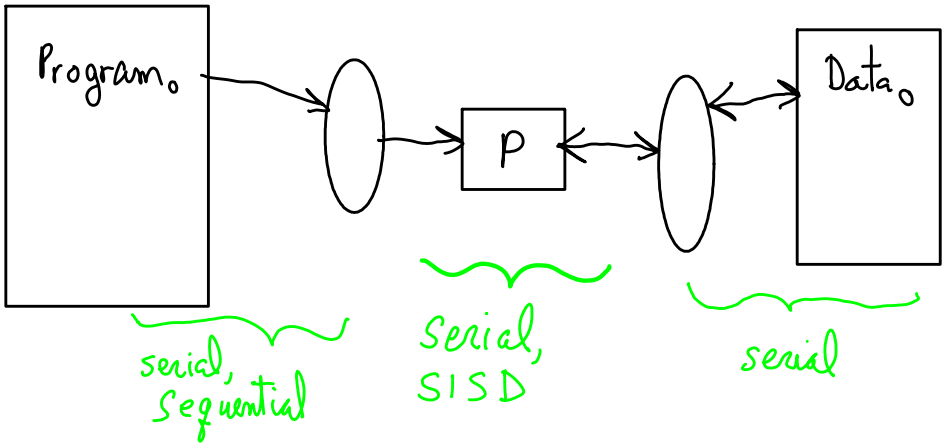
MISD

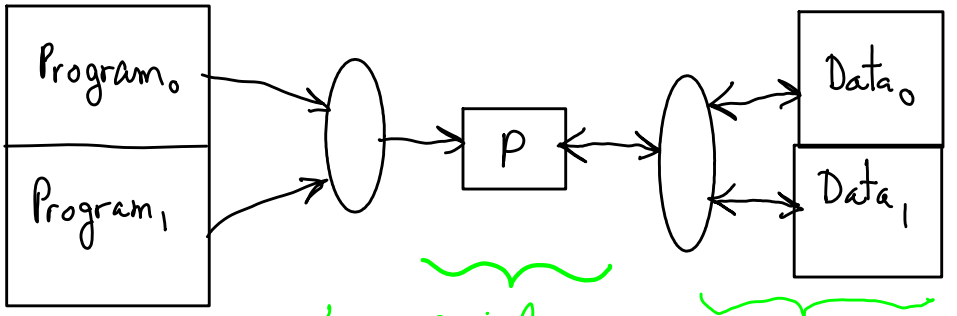
Multi-Instruction, Single Data

(instruction parallel)

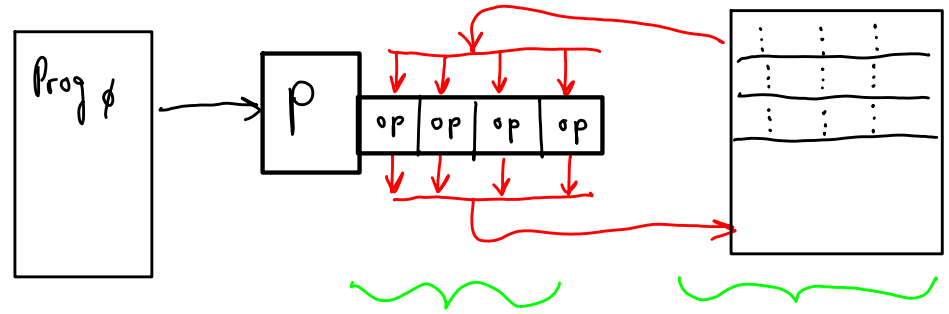


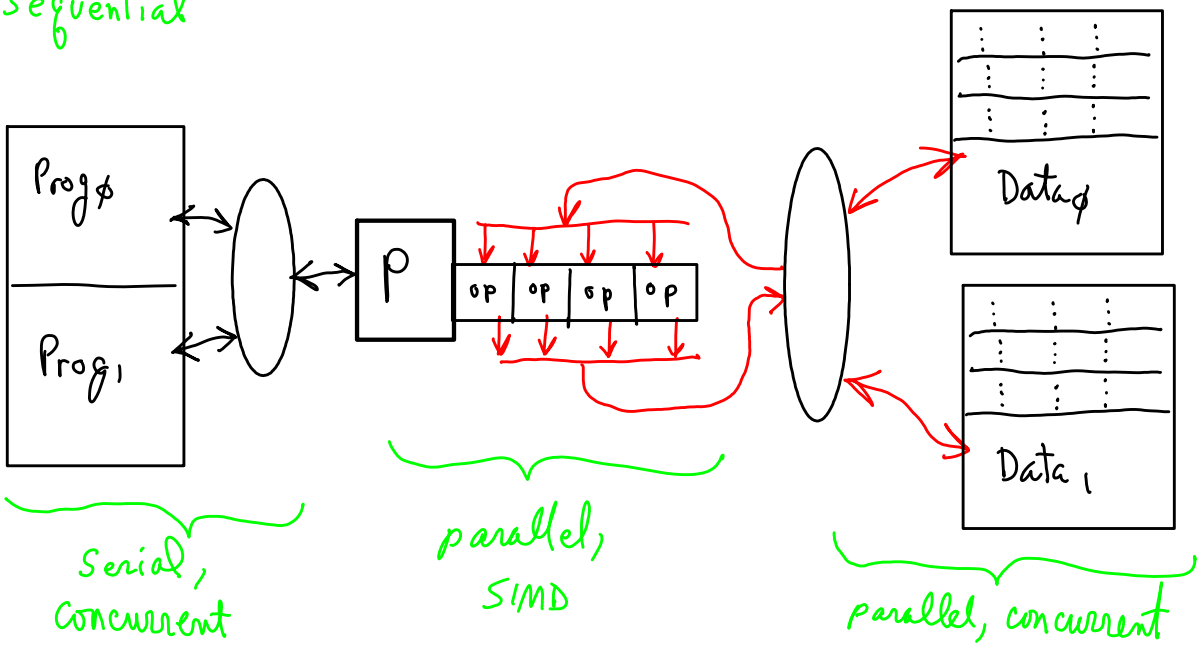
SISD


Program$_0$
P
Data$_0$
serial, sequential
Serial, SISD
serial

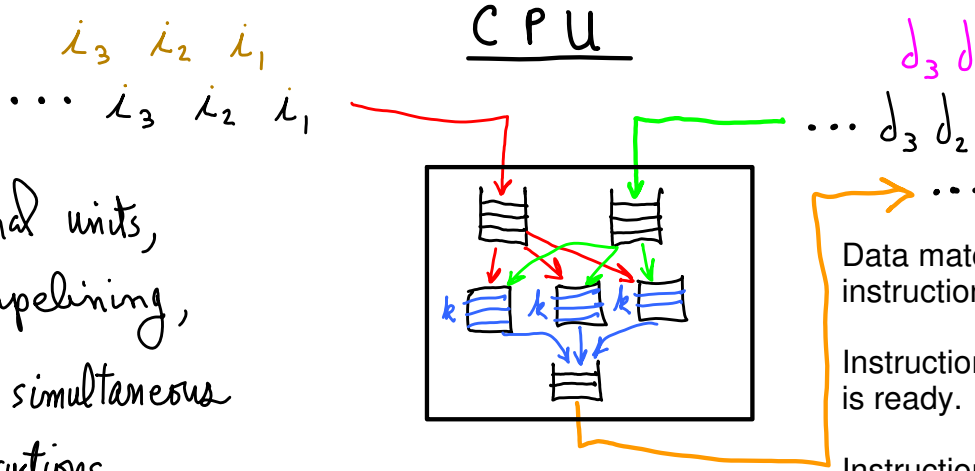

Program$_0$
Program$_1$
P
Data$_0$
Data$_1$
serial, concurrent
Serial, SISD
serial, concurrent

SIMD


Prog$_\phi$
P
op
op
op
op
serial, sequential
parallel, SIMD
parallel, sequential


Prog$_\phi$
Prog$_1$
P
op
op
op
op
Data$_\phi$
Data$_1$
Serial, Concurrent
parallel, SIMD
parallel, concurrent

ILP



n functional units,
k-level pipelining,
$\Rightarrow (n \cdot k)$ simultaneous instructions

+ speculative execution
+ speculative prefetch

Data matched on the fly with instructions (forwarding, etc).

Instructions queued until data is ready.

Instruction schedules reorder.

Commits (state change) are done to preserve semantics.

Nullifying (speculation)

Speed up

W_{real} = output, not computations

$V_{real} = W_{real} / T$

$$S_{2-1} = \frac{V_2}{V_1} = \frac{W_2 / T_2}{W_1 / T_1}$$

$$W_2 = W_1 \Rightarrow S_{2-1} = \frac{T_1}{T_2} = \frac{T_{fixed} + T_{improvable}}{T_{fixed} + T_{improvable} / S_{improvement}}$$

Speed up clock : $T_{fixed} = 0 \longrightarrow S = S_{clock}$

more Pipeline stages (k)

$CR_{new} = k \cdot CR_{old}$, $S_{clk} = k$

more functional units (n)

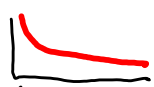
$IPC_{new} = n \cdot IPC_{old}$, $S_{IPC} = n$

$T_{fixed} \approx (1-f)T$: $S \cong \frac{(1-f)T + fT}{(1-f)T + fT/(S_{clk} \cdot S_{IPC})} = \frac{1}{(1-f) + \frac{f}{kn}} = \frac{kn}{kng + (1-g)}$ $\quad g = (1-f)$

$g \sim \frac{1}{kn} \longrightarrow \frac{kn}{1+1}$

(Speculation, forwarding, caching) $\overset{f}{\Leftrightarrow}$ (exposed latency, unused units, pipe + cache flushing/conflicts, pipe bubbles)

BUT  (W/cm^2, device speed)

 (usability)

Scaling

assume: $W_1 = W_2 = W = W_{fixed} + W_{improvable}$

$= (1-f)W + fW$

$S_f = n$ { M_2 is n times faster doing $W_{improvable} = fW$

$V = V_{fixed} + V_{improvable}$ for M_1

$V = V_{fixed}$, $V_{improvable} = V/S_f$ for M_2

$$S_{2-1} = \frac{V_2}{V_1} = \frac{W_2/T_2}{W_1/T_1} = \frac{T_1}{T_2}$$

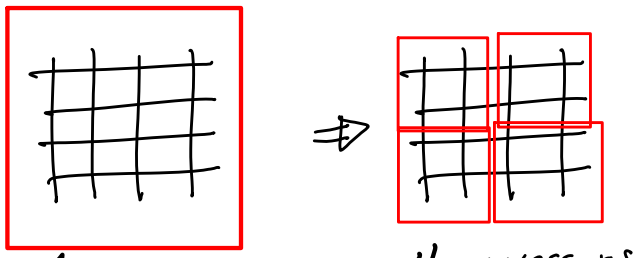
$$= \frac{(1-f)W/V + fW/V}{(1-f)W/V + fW/(S_f V)}$$

$$= \frac{1}{(1-f) + f/n}$$

$$= \frac{n}{n(1-f)+f} \xrightarrow{f \to 1} \frac{n}{0+1}$$

E.G.

grid computation, 10% serial



1 processor ⇒ 4 processors

$$S = \frac{4}{4(1-0.9)+0.9} = \frac{4}{1.3} \approx \frac{4}{4/3} = 3 < 4$$

Caveat:
serial work for M1 < serial work for M4 (coordination, dividing data)

$\Rightarrow W_1 = W_{serial} + W_{parallel} \Rightarrow T_1 = W/V$

$W_2 = W_{serial} + h \cdot W_{parallel} + W_{parallel}$

$$\Rightarrow T_2 = \frac{W_{serial} + h\,W_{parallel}}{V} + \frac{W_{parallel}}{V \cdot S_n}$$

$$= \frac{(1-f)W + h\,fW}{V} + \frac{fW}{V \cdot n} \Rightarrow S_{2-1} = \frac{T_1}{T_2} = \frac{1}{(1-f) + h + f/n}$$

$h = h(n)$

Here h contributes to the serial portion. Could contribute to parallel part.

STRONG scaling

But suppose $h(n) < 0$? $\quad h \in (-1, 0]$

Less serial work needed w/ n processors

$$S_{2-1} = \frac{1}{(1-f+h) + f/n} \xrightarrow{h \to (f-1)} n/f = \Theta(n)$$

or

$W_{parallel} \to fW + hW$ Less parallel work needed w/ n processors

$$S_{2-1} = \frac{1}{(1-f) + (f+h)/n} \xrightarrow{h \to -f} \frac{1}{(1-f)}$$

weak scaling

More processors ==> Do a larger problem

$$W = W_{fixed} + n \cdot W_{improvable}$$

(or generally $r(n) \cdot W_{improvable}$)

e.g. $r(n) = \Theta(n)$

$$T_2 = W_{fixed}/V + \frac{n \cdot W_{improvable}}{V}$$

$$S_{1-2} = \frac{W_{fixed}/V + \frac{n \cdot W_{improvable}}{V}}{W_{fixed}/V + \frac{n \cdot W_{improvable}}{n \cdot V}} = \frac{(1-f) + nf}{(1-f) + f} = 1 + f(n-1)$$

$$= \Theta(n)$$

for our previous example, 4x larger grid

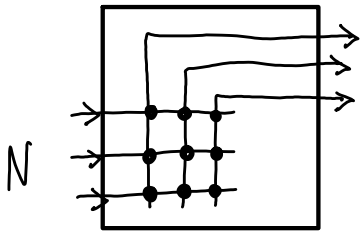
$= 1 + 0.9(3) = 3.7$

In general, all above effects can play a part (+ or -).

--- serial work for coordination, initialization
--- parallel work for parallel portion, problem remapping
--- communication overhead and resources
--- memory and cache bandwidth effects
--- cache coherency

interconnects switched

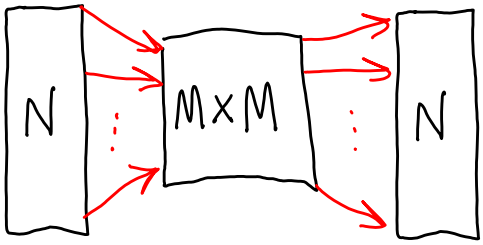
N x N Crossbar switch



route
- any 1–1 map
- any (N-k) x N * fan-out
- any N x (N-k) * fan-in

* collisionless (buffering ?)

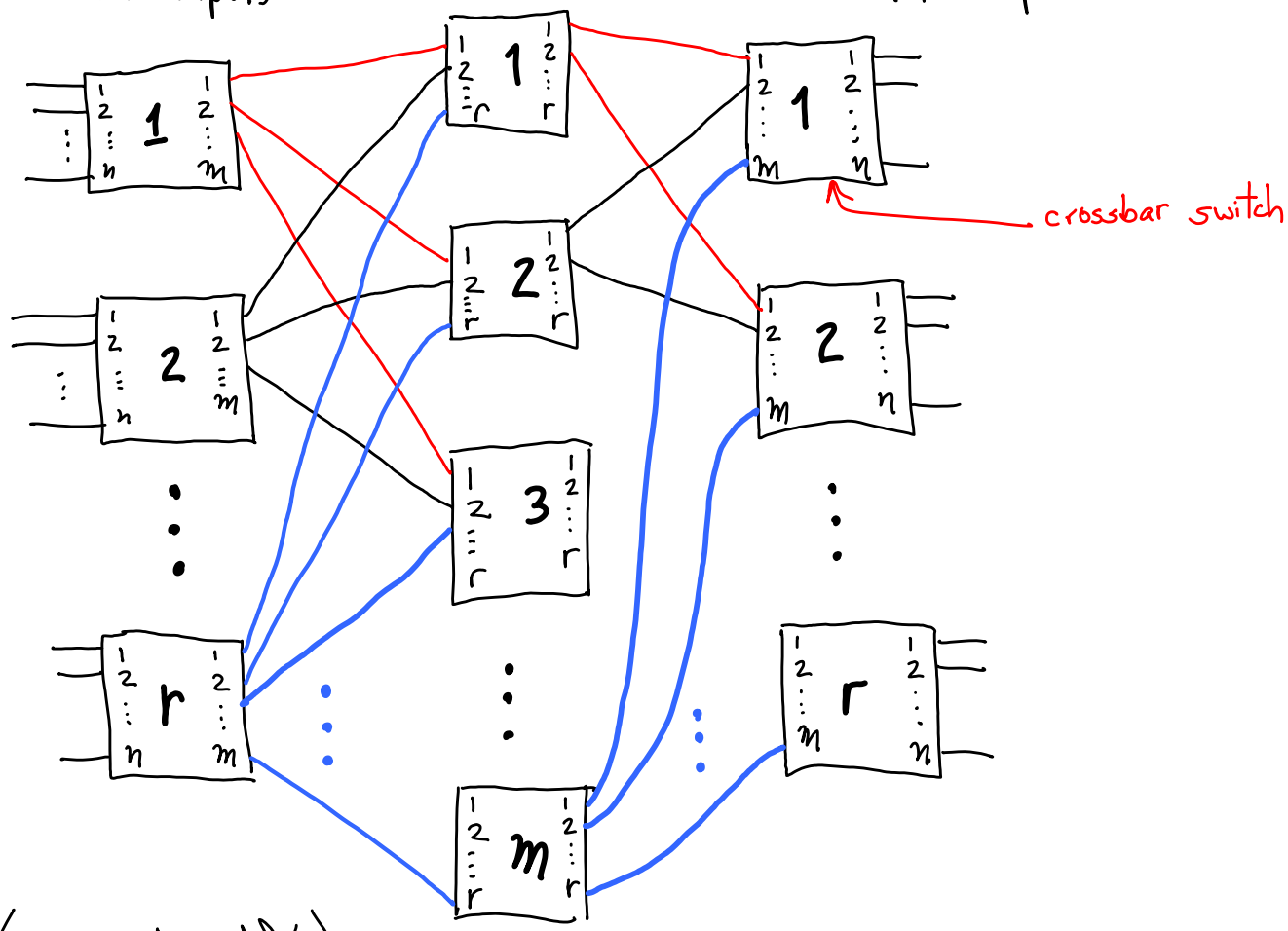
Clos



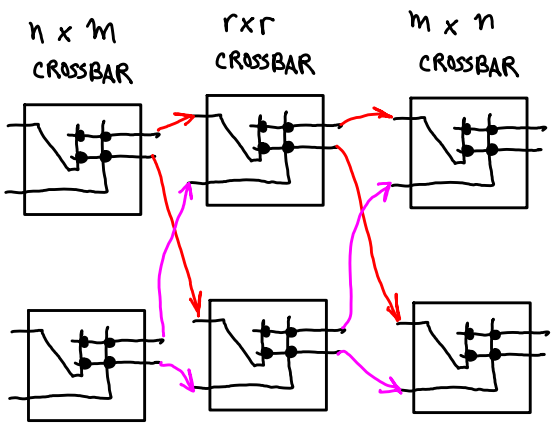
N = n·r inputs

M = r·m

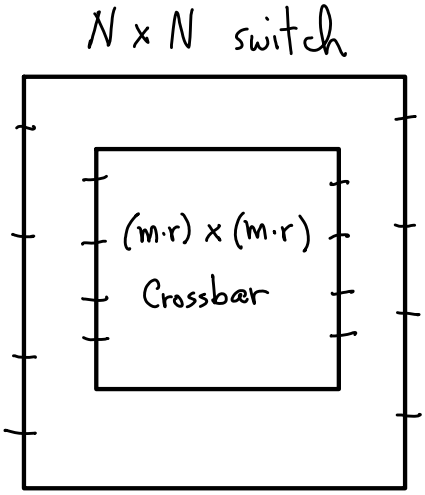
N = n·r outputs



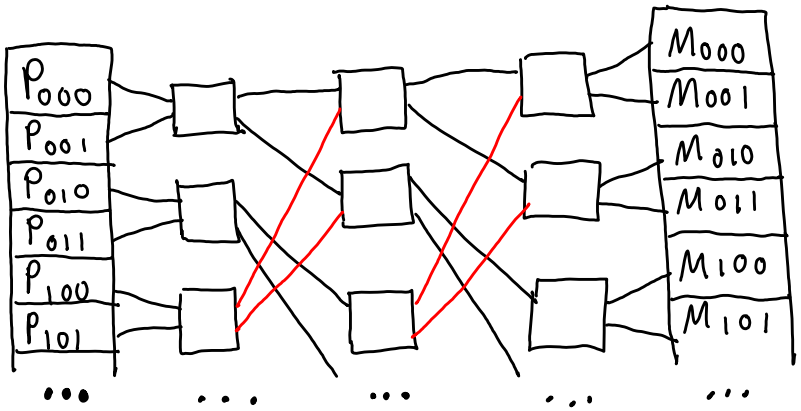
(recursive defn)


n x m
CROSSBAR
r x r
CROSSBAR
m x n
CROSSBAR

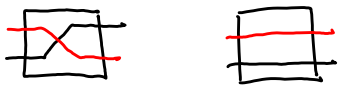
=


N x N switch
(m·r) x (m·r)
Crossbar

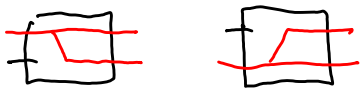
Ω 8×4 interconnect



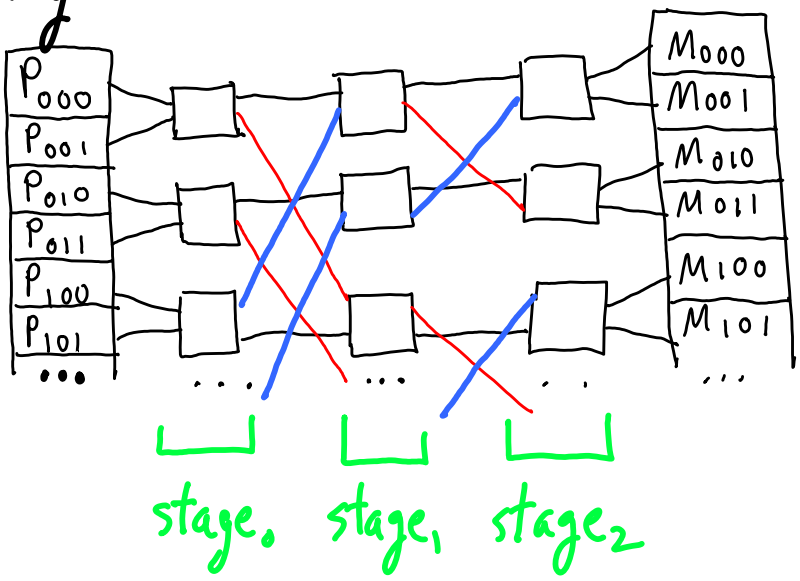
switch ⇒ blocking



also, can broadcast:



Butterfly 8×4



(Banyan)

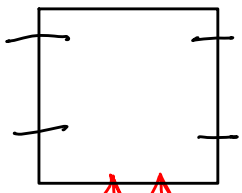
n Ps

n Ms

$\log n$ stages, @ $n/2$

$\log n$ address bits

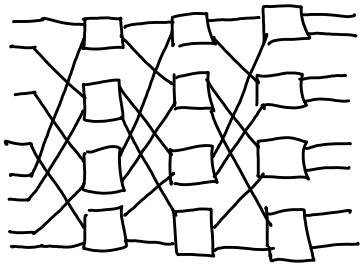
Switch at k^{th} stage:



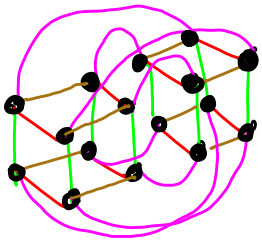
$addr_0$ | $a_{\log n}\ a_{\log n-1} \dots a_k \dots a_1 a_0$

$addr_1$ | $a_{\log n}\ a_{\log n-1} \dots a_k \dots a_1 a_0$

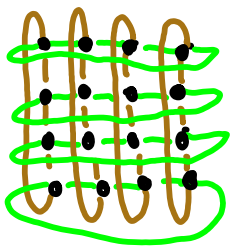
shuffle - exchange (perfect card shuffle)



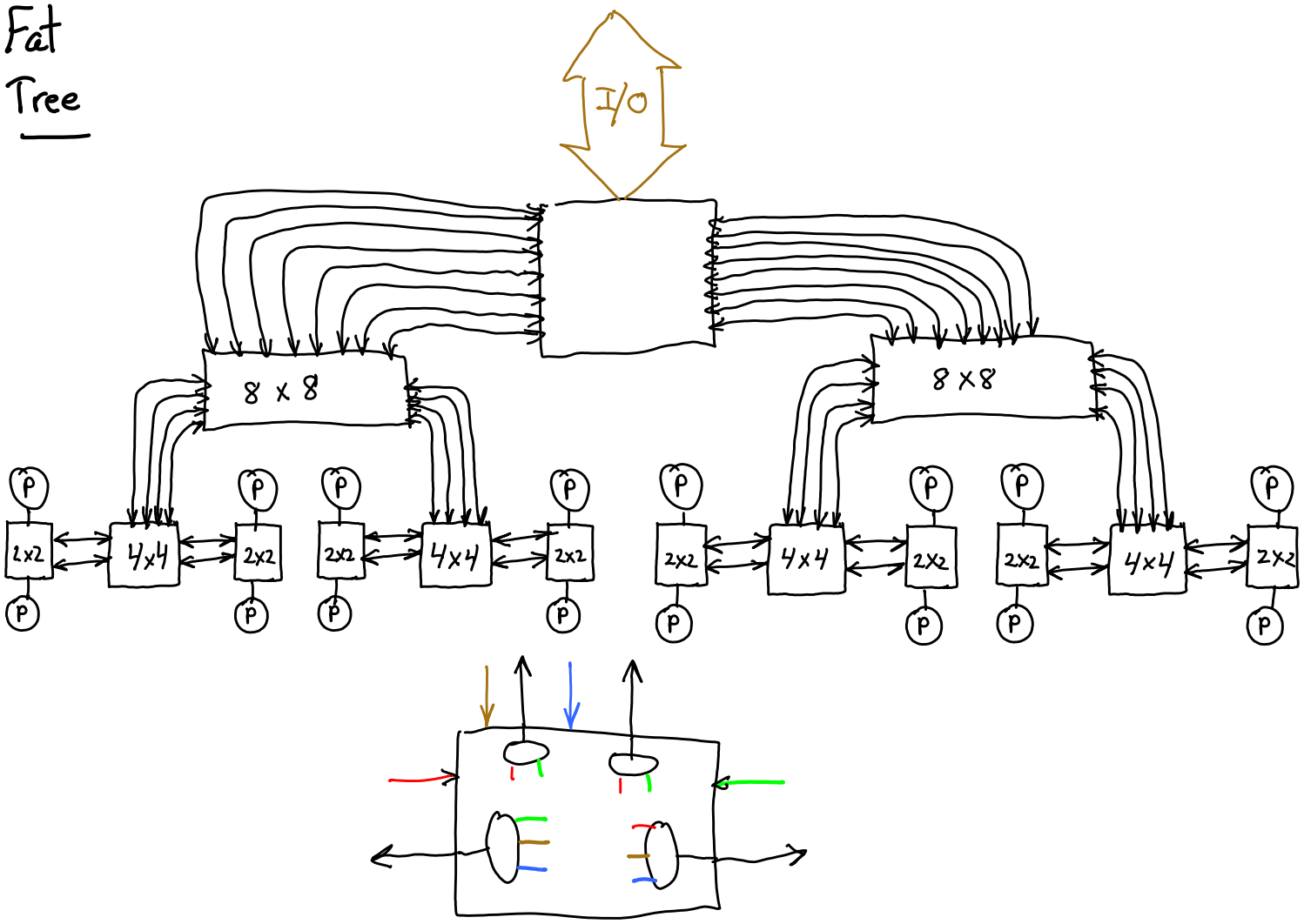
hypercube



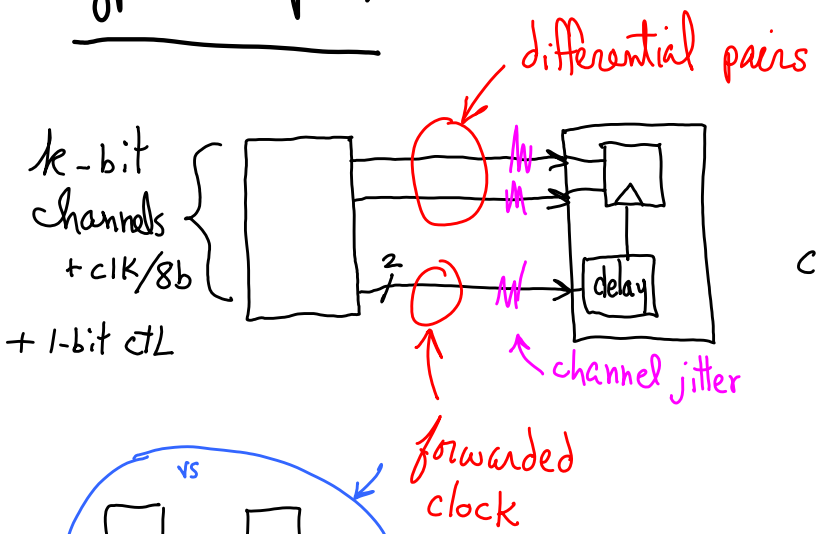
Torus

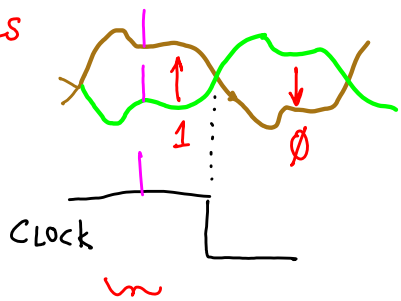


Fat Tree

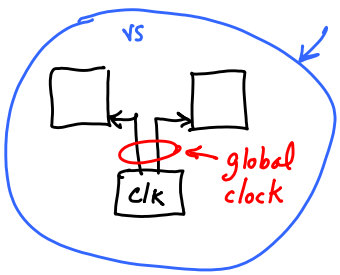


HyperTransport

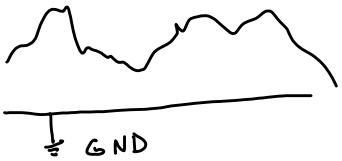




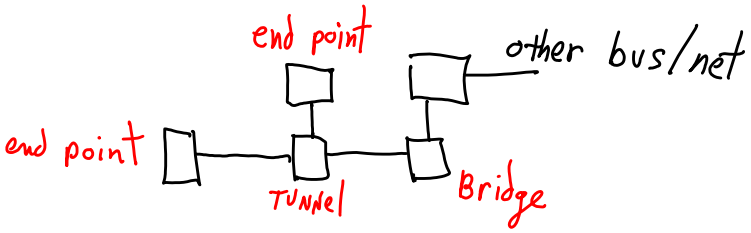
high-speed signals are bouncy (jitter), and corrupted by other signals.



vs.



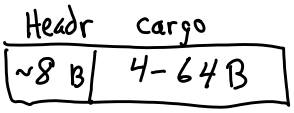
Detecting whether signal is up or down is difficult.



daisy chained, point-to-point

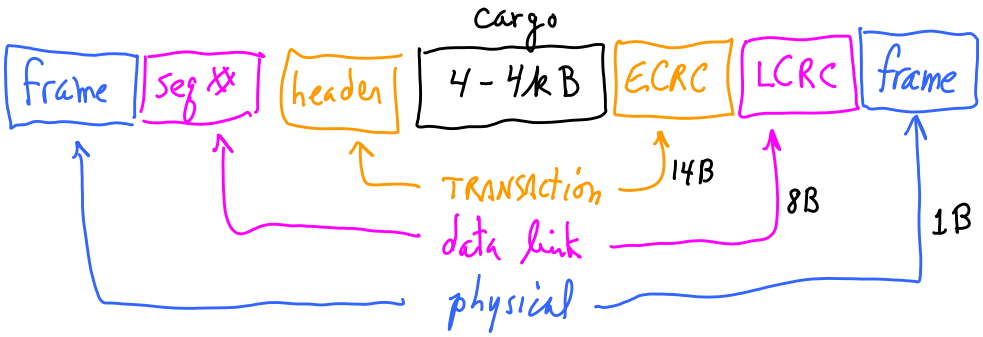
packets { 4-64 B data | 4-8 B CTL }

Packet overhead



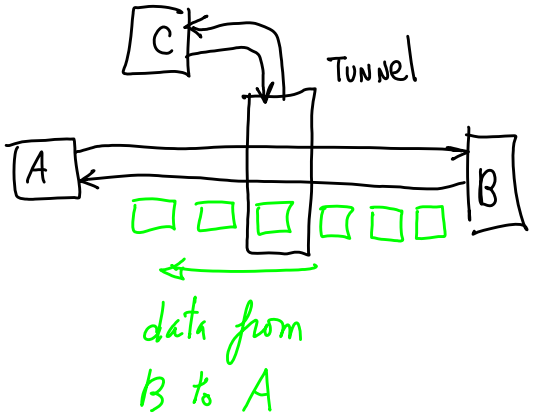
HT

Transaction / data link / physical



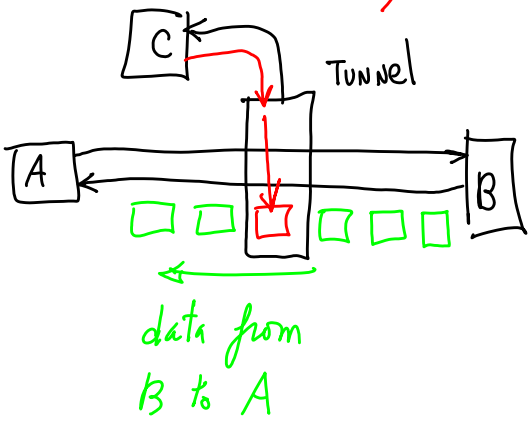
PCIe ~24 B overhead

Priority / interleaving

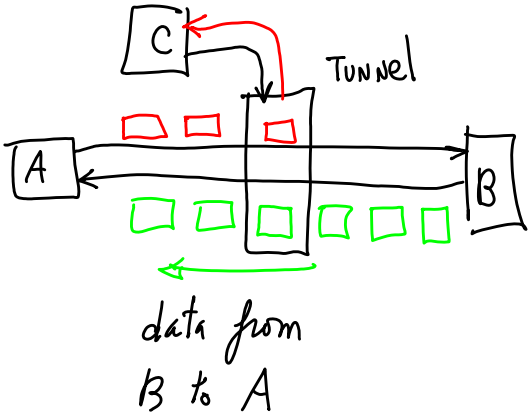


Links in pairs

Priority interleave

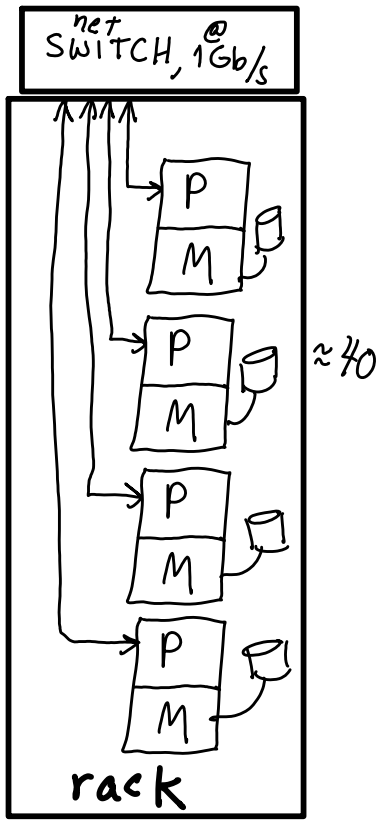


data from A to C



⇒ Low Latency response

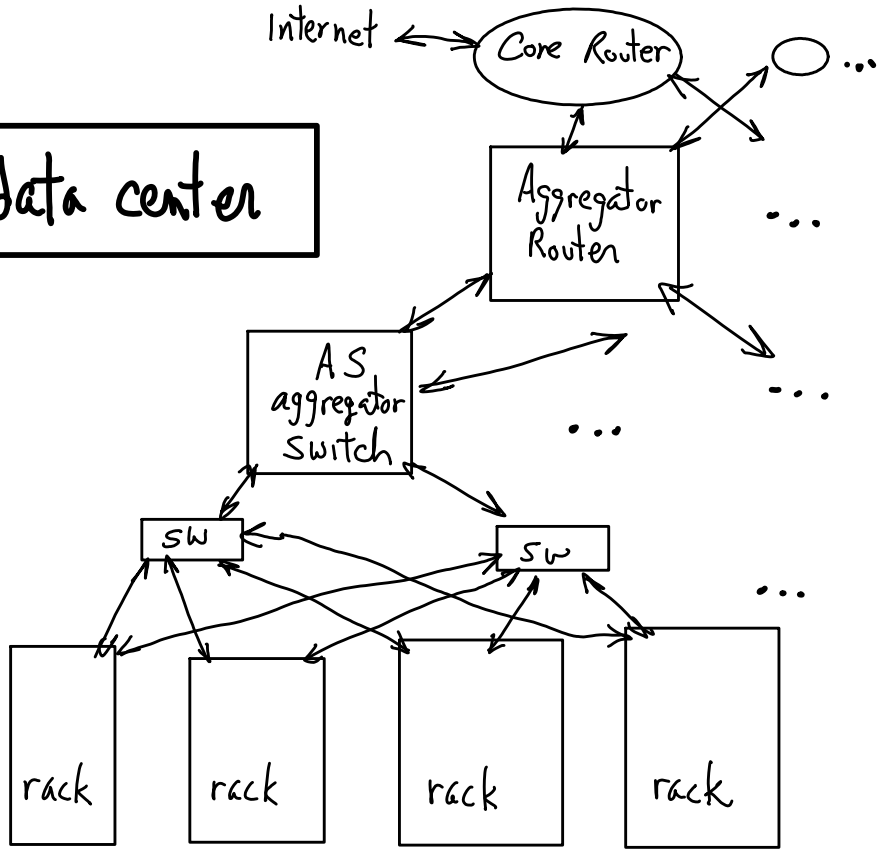
Cluster



non-shared memory, message passing

Thread parallelism
Process parallelism
Task parallelism

data center

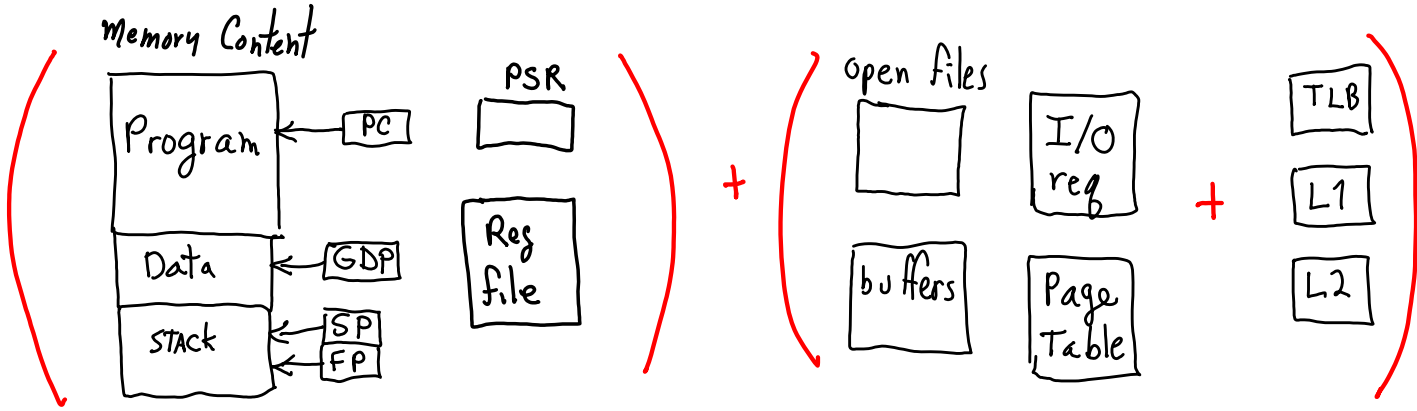


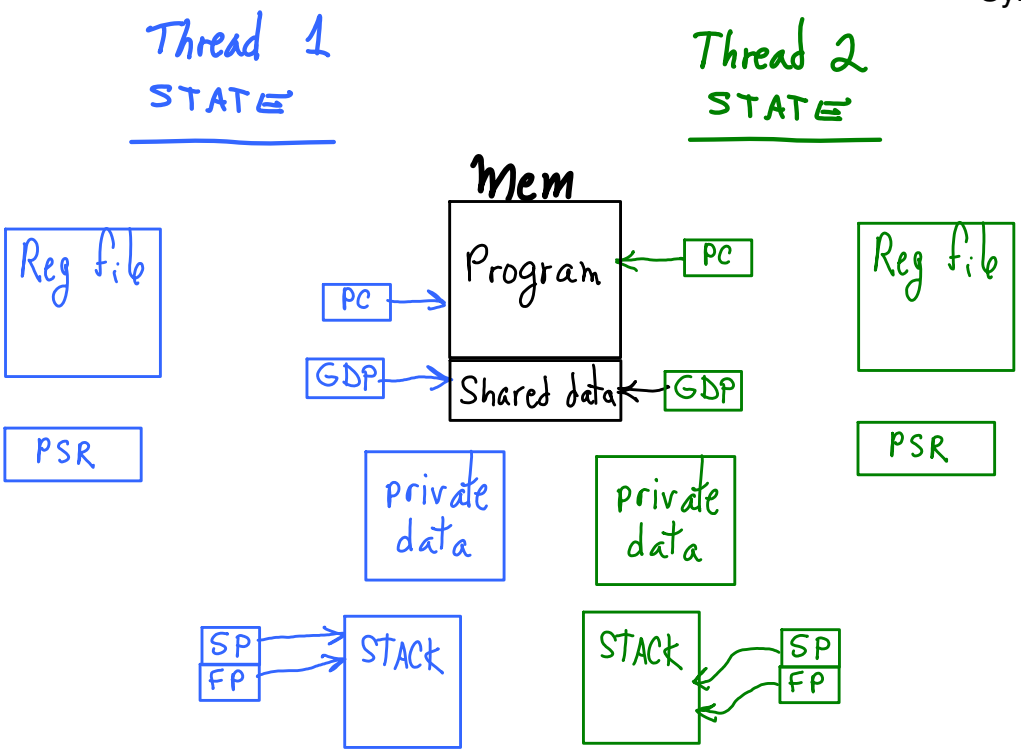
→ difficult algorithm-to-program paradigm
→ slow communication
→ multiple copies of OS, software
→ multiple administration
→ middle ware overhead
→ hot swappable
→ expandable
→ commodity components
→ virtualization, migration, redundancy

Threads

Process context switch ≈ 1M instructions

Process State





Switch threads

Software:
copy state-from
Copy state-to
load PSR, PC, SP, FP

TLB, L1, L2 ok.
PT ok (mostly?)
File tables are ok?
Buffers? I/O reg.?

Software threads

Switching thread Context

In OS code, or user code:

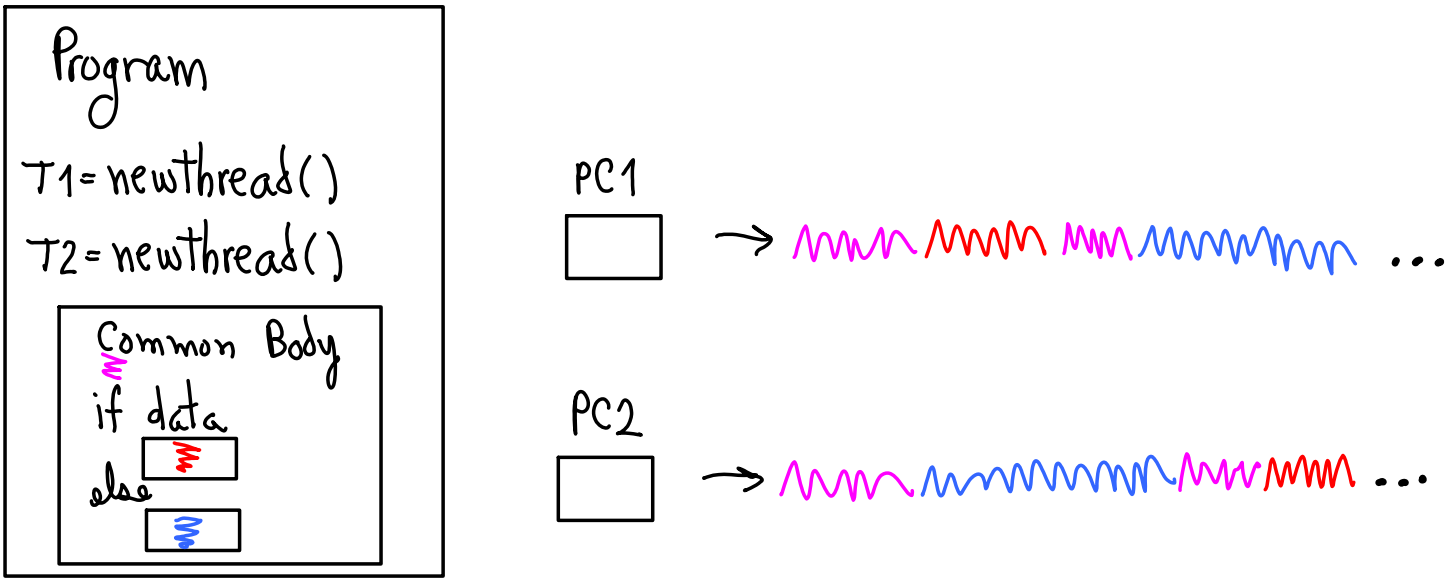
- save state T1
- load state T2

run T2 until

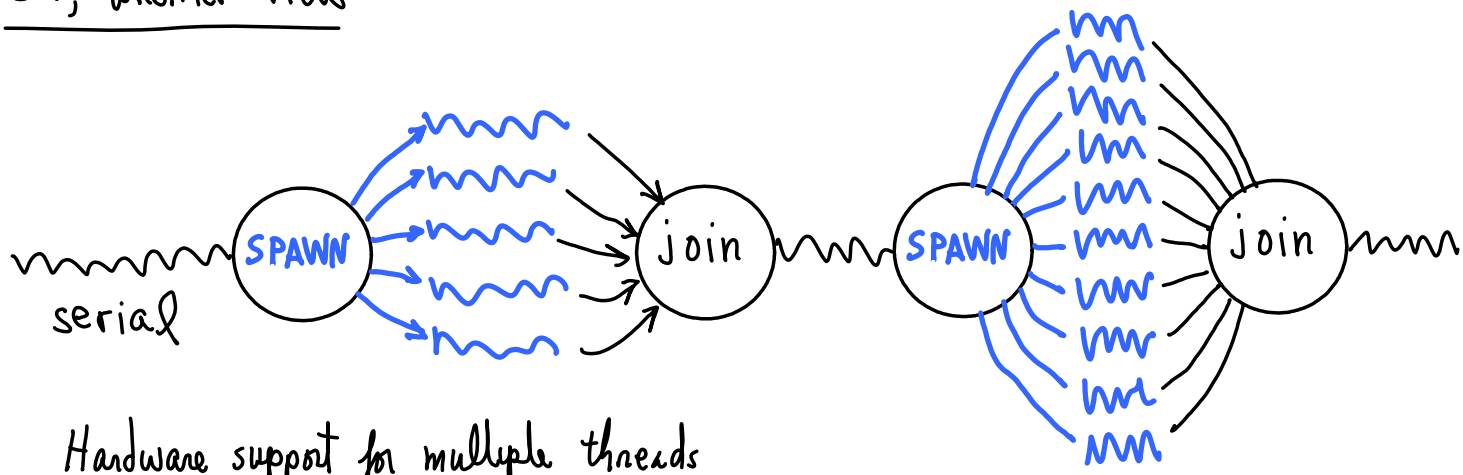
- HW timer
- IO event sleep
- cooperative hand-off

Switch context

Instruction streams



OR, another view



Hardware support for multiple threads

ALL THREADS from SAME PROCESS

--- Duplicate and switch: PC, PSR, RegFile, Stack, Private data

--- Copy/Save/Restore state

--- Shadow registers, renaming

--- TLB content (separate page tables? or shared?)
  --- hardware switch
  --- TID, thread ID labeled

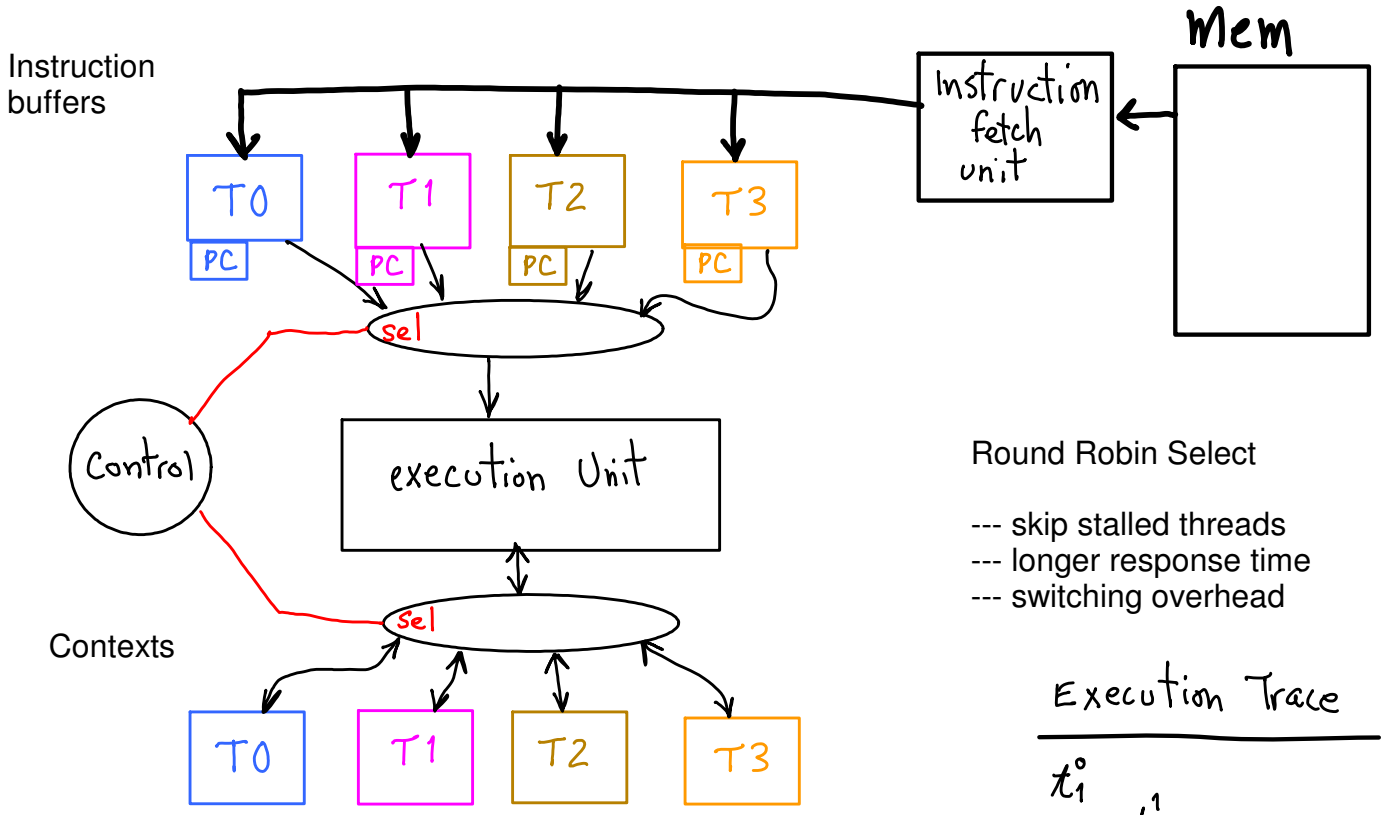
--- MULTIPLE THREADS from MULTIPLE PROCESSES

--- PID + TID

--- Larger state to consider (page tables, file and IO tables and buffers)

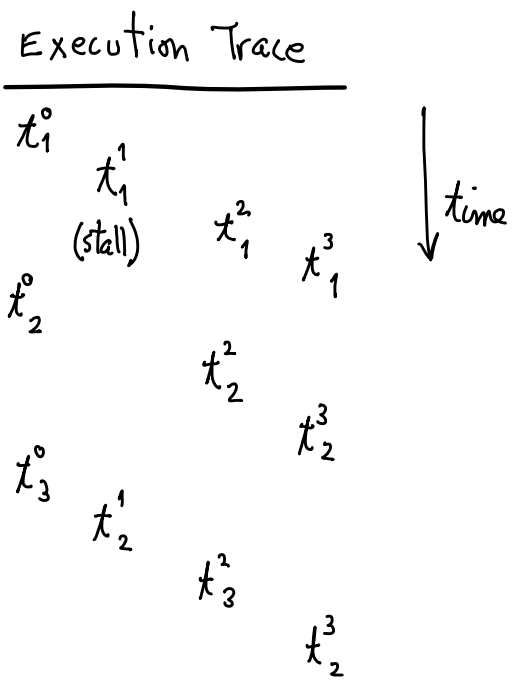
OS policy not known to HW designer?

Fine-grained multi-threading

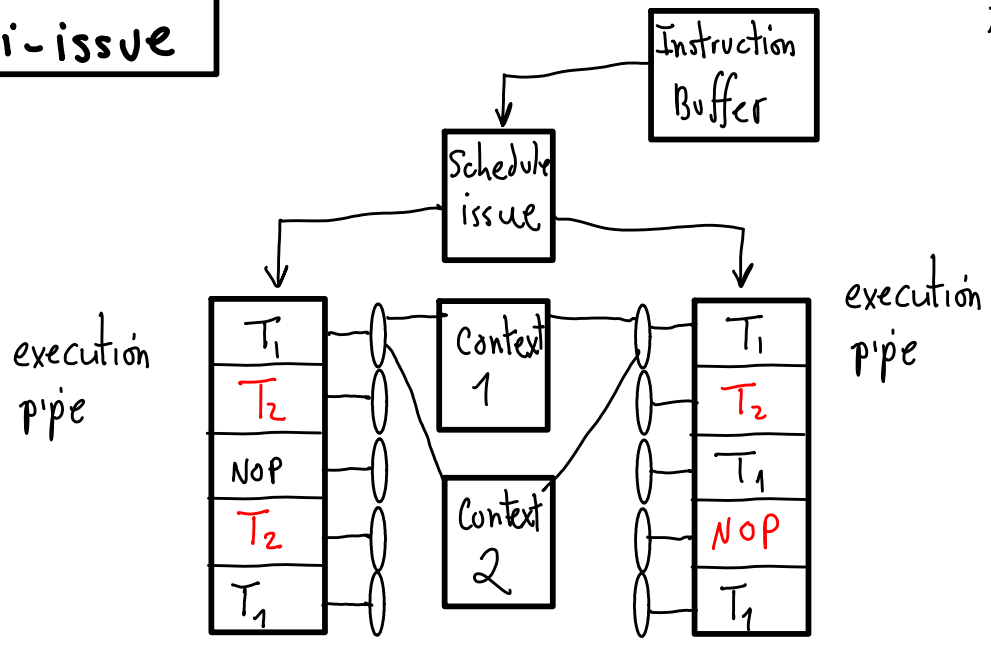


Round Robin Select

--- skip stalled threads
--- longer response time
--- switching overhead



multi-issue



Coarse Grained

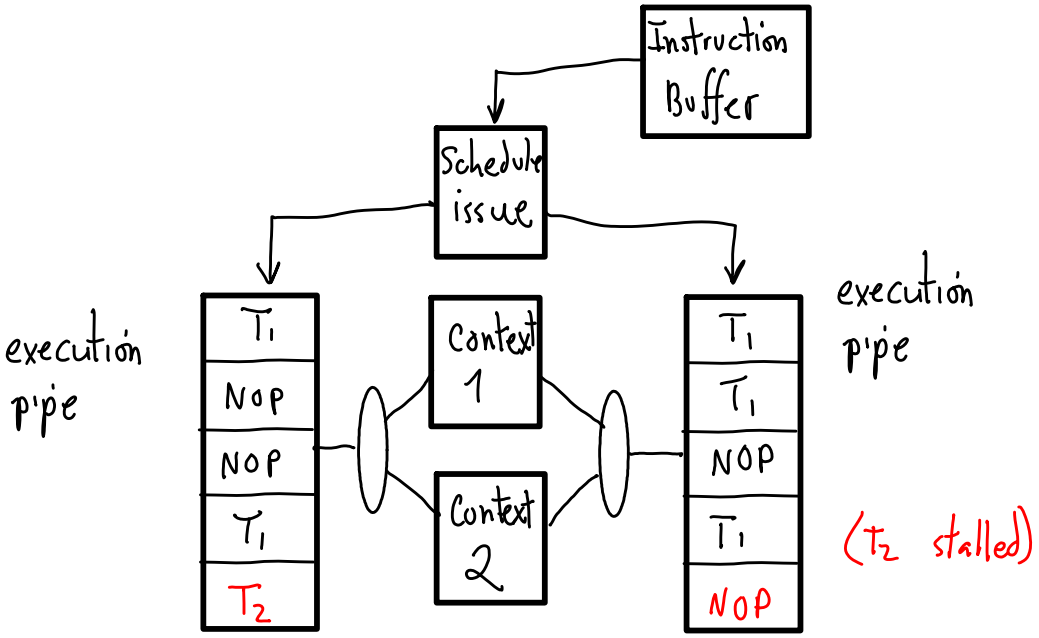
Same HW, but schedule same thread until stalled.

Execution Trace

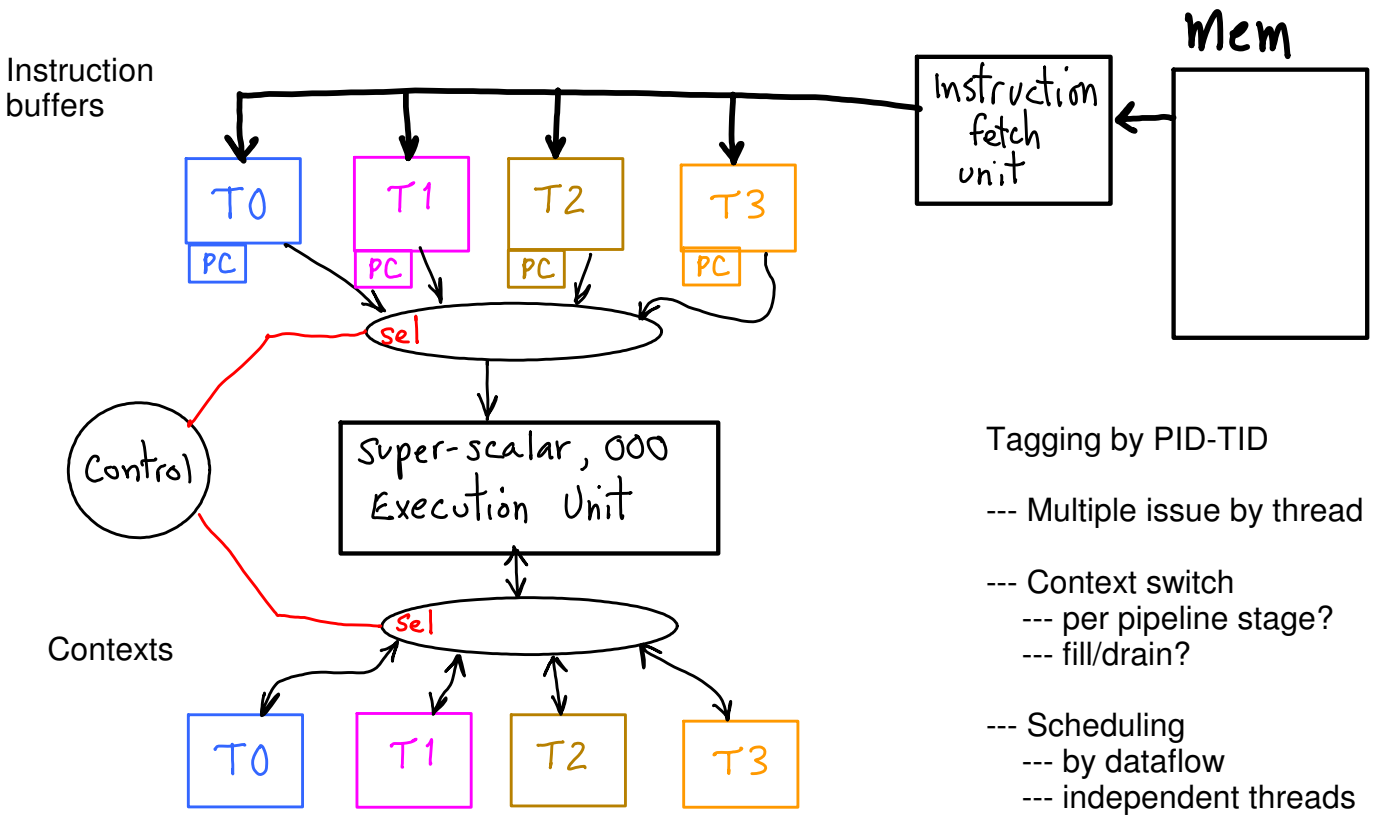
t_1^0
t_2^0
t_3^0
(stall) t_1^1
(stall) t_1^2
t_2^2
t_3^2

--- Faster response time
--- lower switching overhead
    --- affords more complex control
    --- faster execution between
--- Starvation now a problem?

Multi-issue



(Simultaneus)(Hyper) Threading



Tagging by PID-TID

--- Multiple issue by thread

--- Context switch
  --- per pipeline stage?
  --- fill/drain?

--- Scheduling
  --- by dataflow
  --- independent threads
    more ==> parallelism

time

T1	T1	T2	T3
T2	T0	T0	NOP
NOP	T1	T1	T1
NOP	T2	T3	T1
T3	T3	T2	T2

← multiple issue
- OOO scheduling across threads
- more filled slots

Single threaded execution

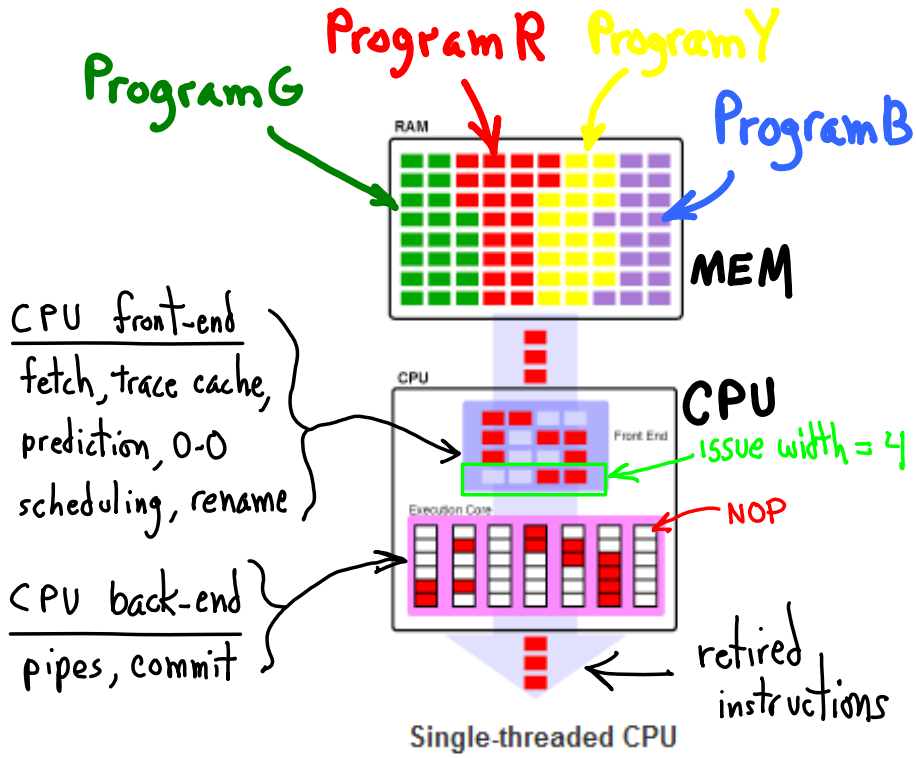
Multi-tasking

---- Multiple concurrent execution
(not simultaneous)
---- Memory shared but separate (virtual)

---- CPU time-multi-plexed
--- cooperatively, pre-emptively, IO

---- Process context switching
drain/fill (pipes, caches, TLB, ...)

---- Extract ILP from single stream
---- Unused issue slots
---- pipeline bubbles/stalls



Single-threaded CPU

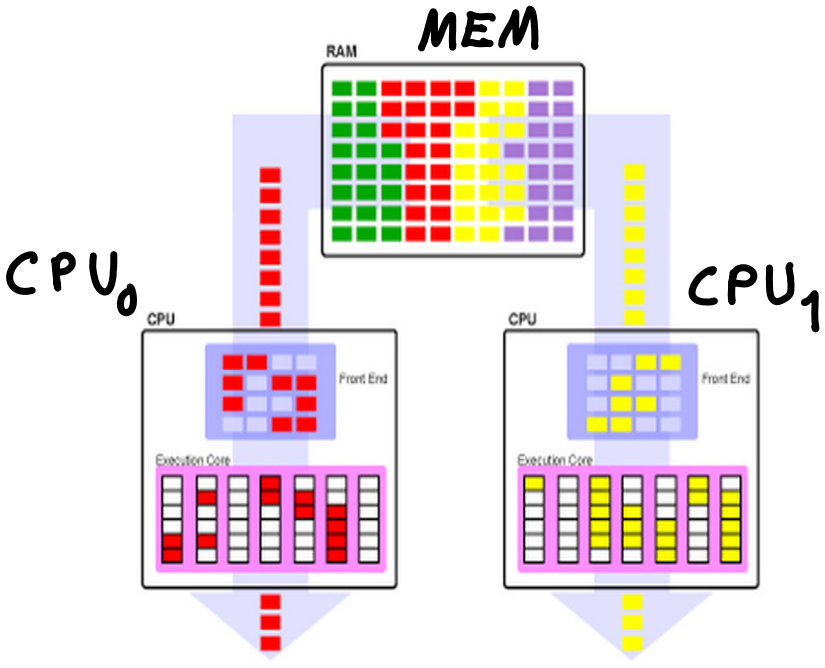
Credits:
Introduction to Multithreading, Superthreading and Hyperthreading
By Jon Stokes

SMP, Symmetric Multi-Processing

---- Context switch per CPU

---- Simultaneous execution
--- multiple programs/processes/threads

---- ILP extracted per process
--- double silicon resources
--- same NOP density
==> Could speedup be > 2?



Single-threaded SMP

Multi-Threaded (Superthreading)

---- Concurrent process scheduling
--- process context switching
--- cooperative, pre-emptive, ...

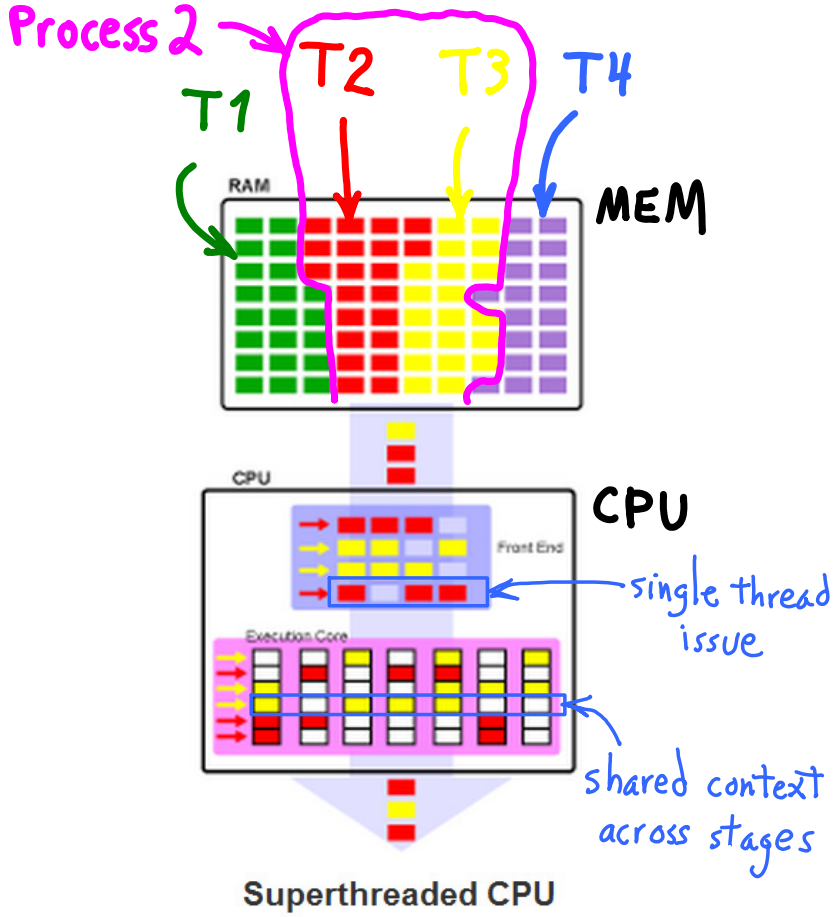
---- Single process, multiple thread execution

---- Time-multiplexed thread scheduling
--- from same thread

---- Instructions issue from single thread

--- thread context switch
-- in HW
-- per stage
-- across stages

---- Execution slots filled
--- due to stalled threads
--- filled from non-stalled threads
--- Lower density of NOPs



Superthreaded CPU

SMT, Simultaneous Multi-(Hyper)-Threading

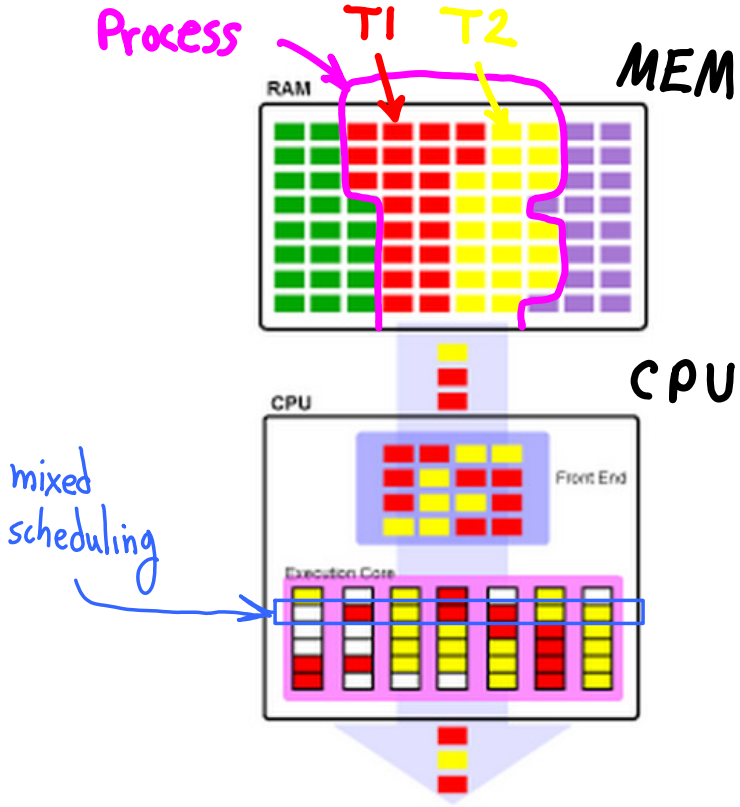
---- Concurrent Processes
--- context switching

---- Thread context switching
--- independently on different pipes
--- issue from multiple threads simultaneously

---- Average ILP = 2.5, empirically
--- max single-thread issue = 4 (here)
--- combined ILP ==> 4

---- Logical Processors == 2

---- Lowest NOP-density



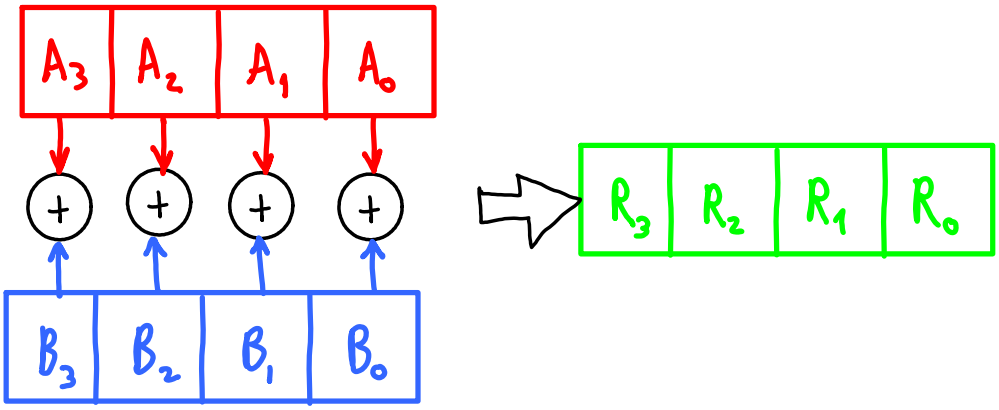
Hyper-threaded CPU

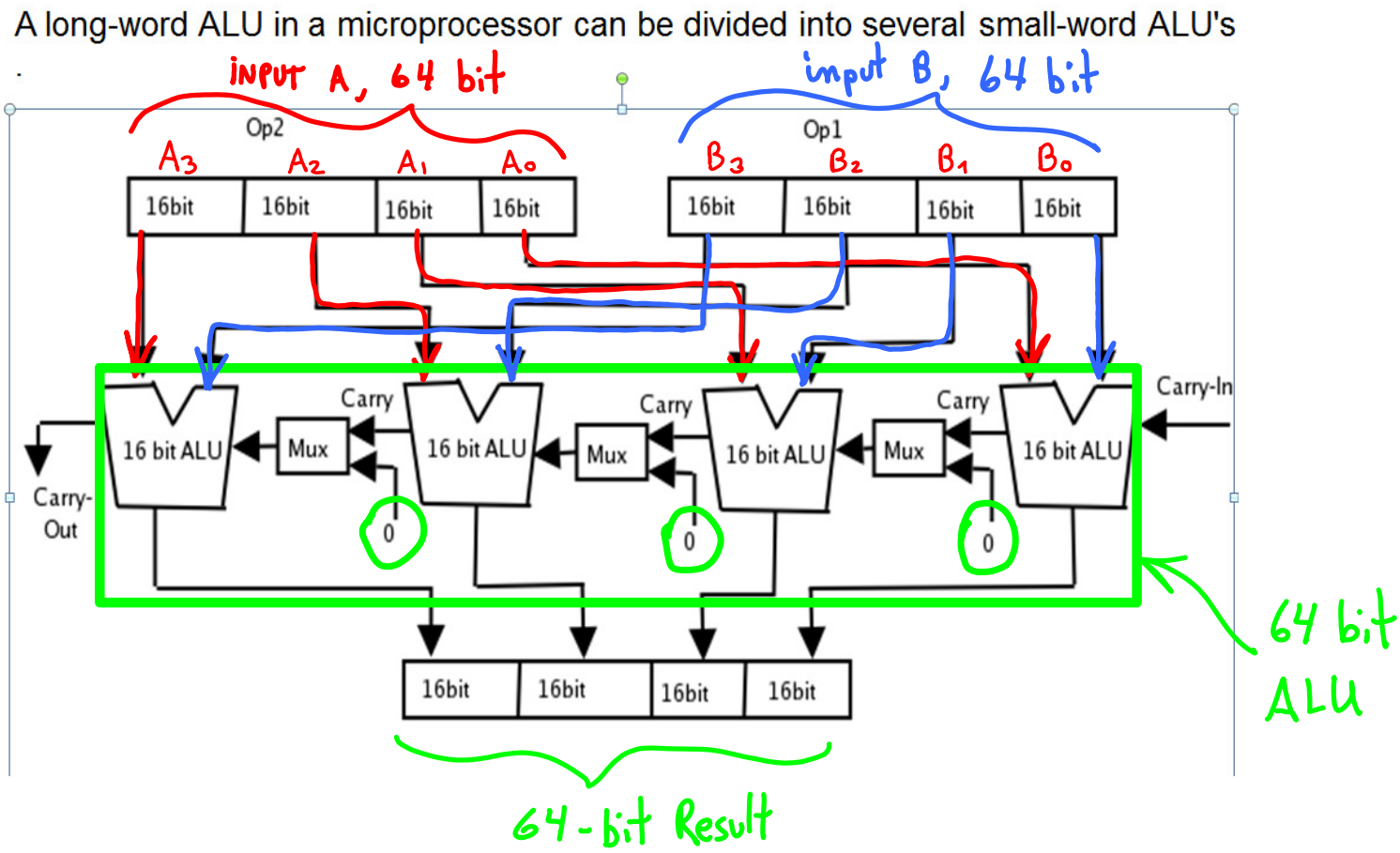
Modification of existing O-O-O CPU
⇒ 10% added cost, $s^{\prime} > 2$

SIMD, cheap

Why not use existing ALU as vector processor?

Vector ADD: R = A + B





what for? 16-bit sound DSP?

Intel MMX ⇒ added to ISA: larger

1) Vectors (more elements)
2) elements (more bits)

Predicated execution

what to do w/ IFs?

Loop (per vector)

if $A_i > B_i$

$R_i \leftarrow A_i + B_i$

else

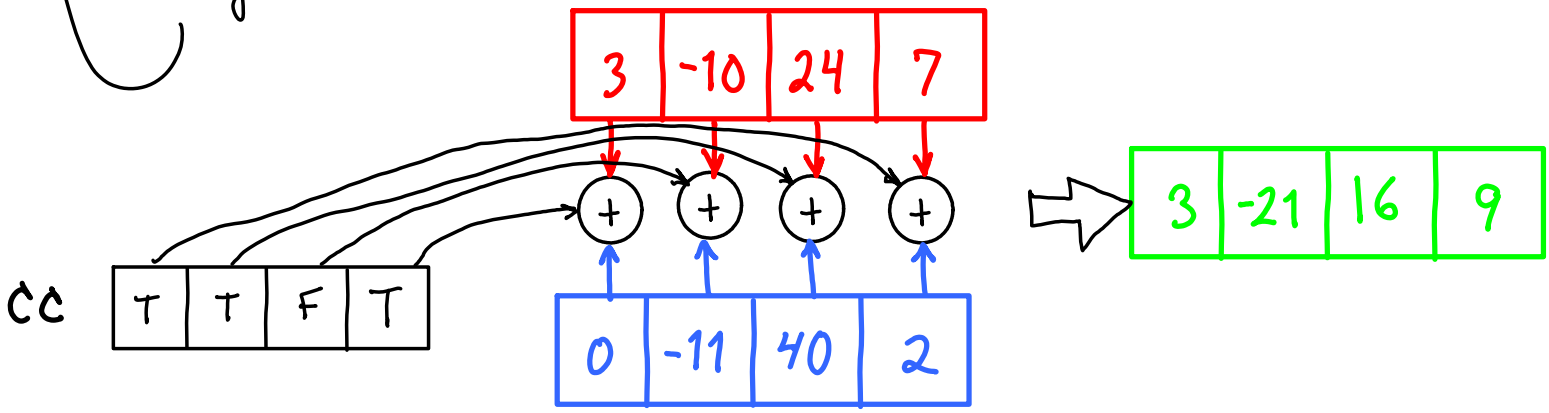
$R_i \leftarrow (-A_i) + B_i$

endif

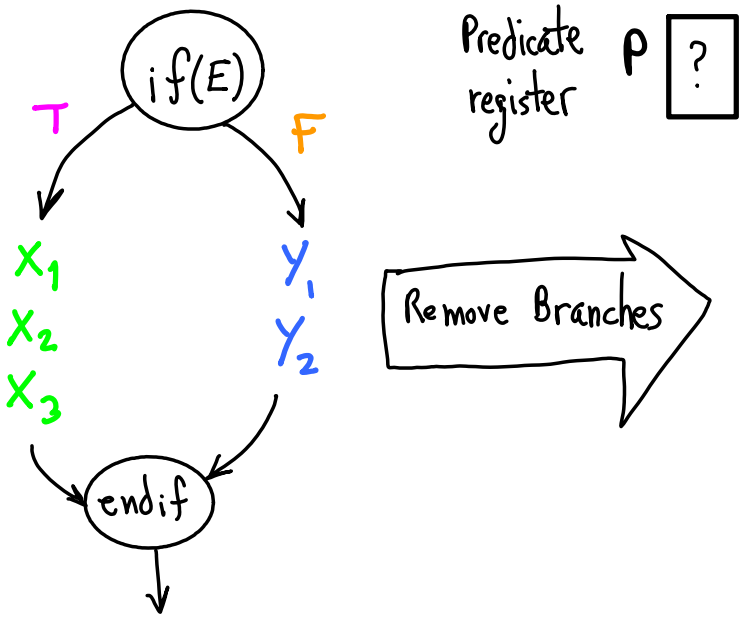
$\Rightarrow$ $(A-B) \Rightarrow CC$ (predicate vector)

CC_3	CC_2	CC_1	CC_0
T	T	F	T

Predicate vector



Predicated execution for non-vector ops



Predicate register p ?

Remove Branches

evaluation of E

Op	SRC_0	SRC_1	P

X_1 T Op (no effect if p = F)

Y_1 F Op (no effect if p = T)

X_1 T Op

Y_2 F Op

X_1 T Op