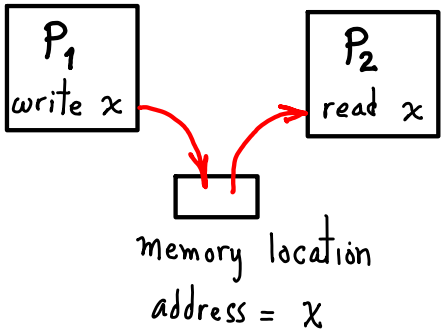
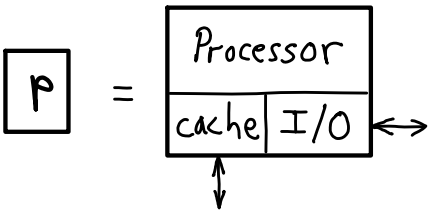
Parallel Systems

common address space PRAM model

multi-processor, shared memory (SMP)

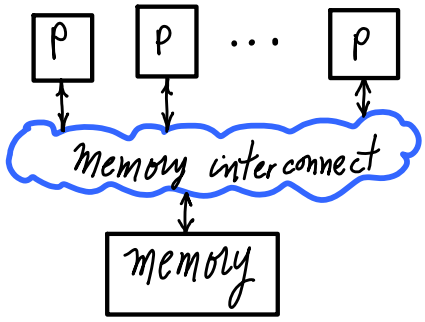


coordination through memory
(can implement messages on top)



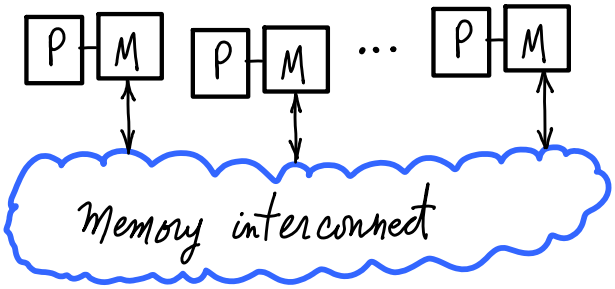
- Uniform access to memory: UMA

--- Global clock
--- ALL Procs communicate at same time
--- ALL data sent/received in one step



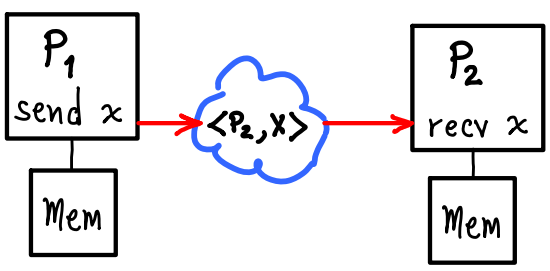
- non-UMA: NUMA

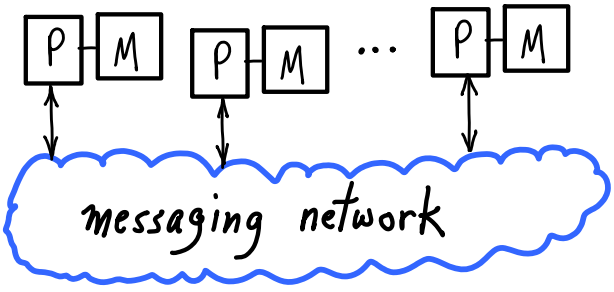
--- Local clocks
--- Memory delays vary
Local: fast
Remote: slow
--- interconnect hierarchy



Non-shared Memory: message passing

coordination through messages

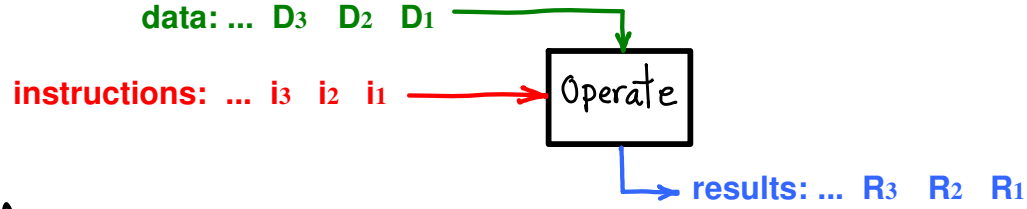




Taxonomy

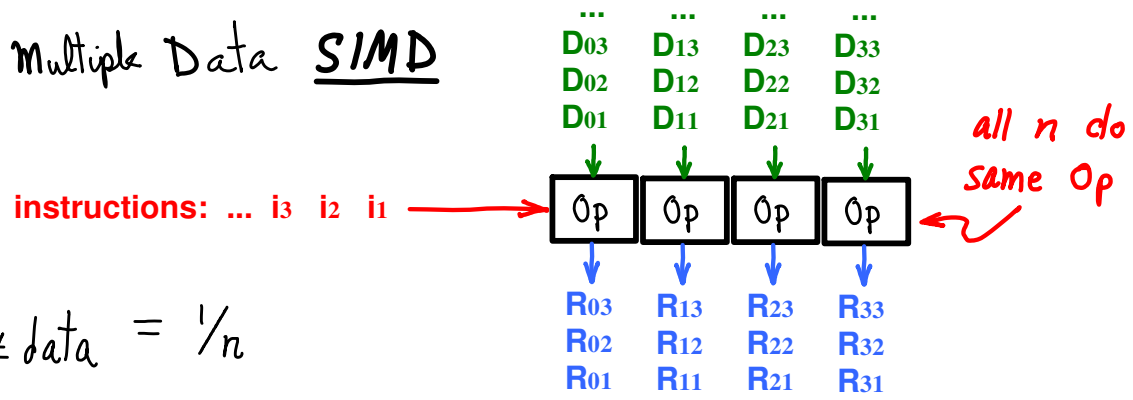
Single Instruction, Single Data STREAM STREAM **SISD**

serial



data-level parallelism

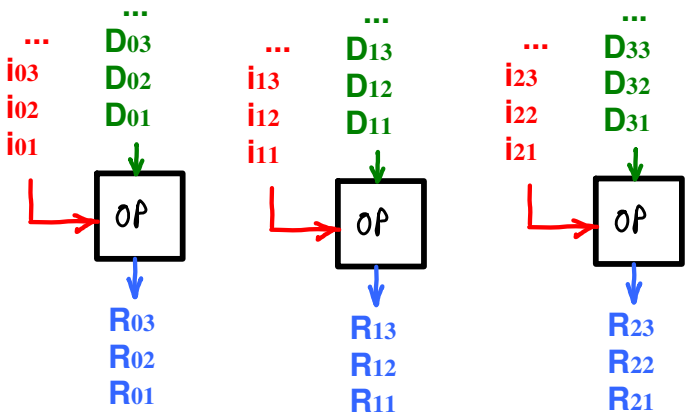
Single Instruction, Multiple Data **SIMD**



$\#\text{instructions} / \#\text{data} = 1/n$

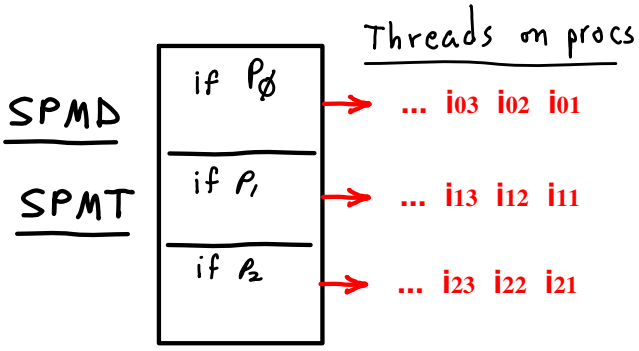
instruction and data parallel

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



SPMD

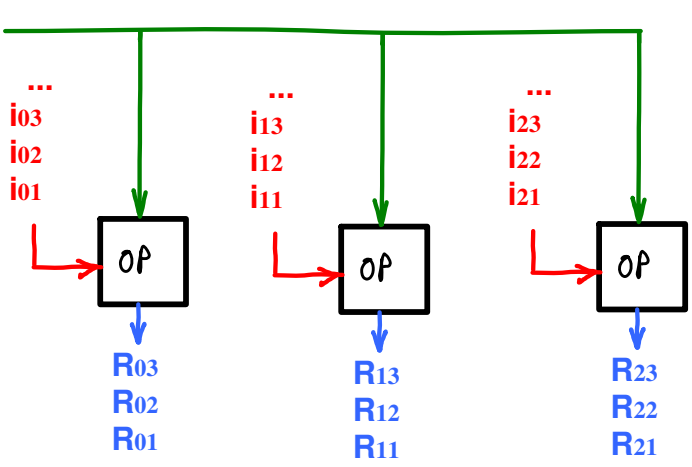
SPMT

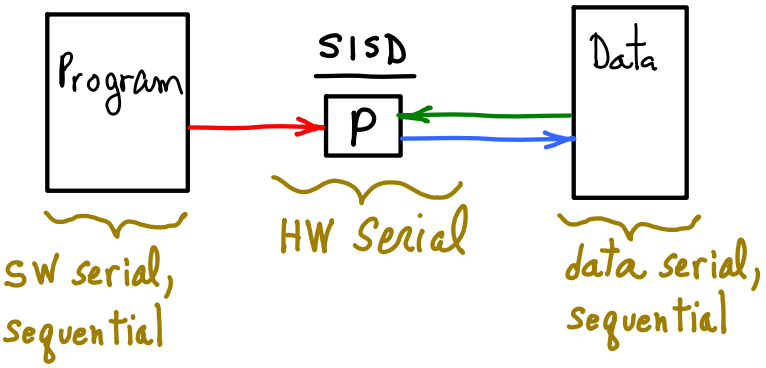


instruction parallel

Multi-Instruction, Single Data

MISD

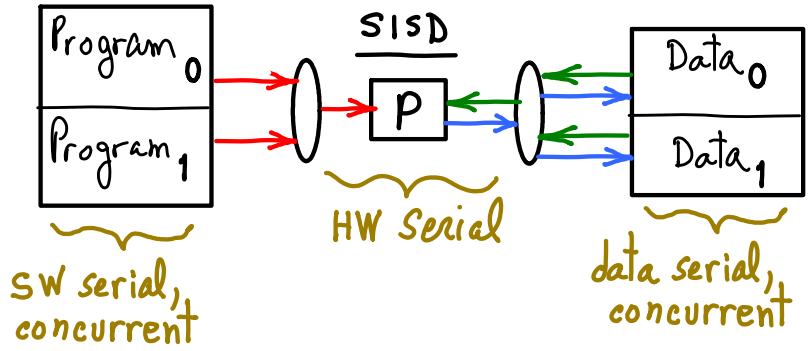




SW serial: single instruction stream
SW sequential: single-process

HW serial: non-parallel execution

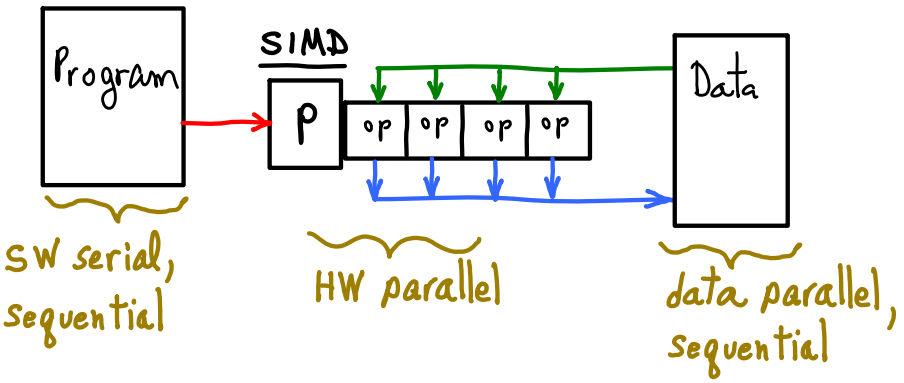
Data serial: single data stream
Data sequential: single-source



SW serial: single instruction stream
SW concurrent: multi-process

HW serial: non-parallel execution

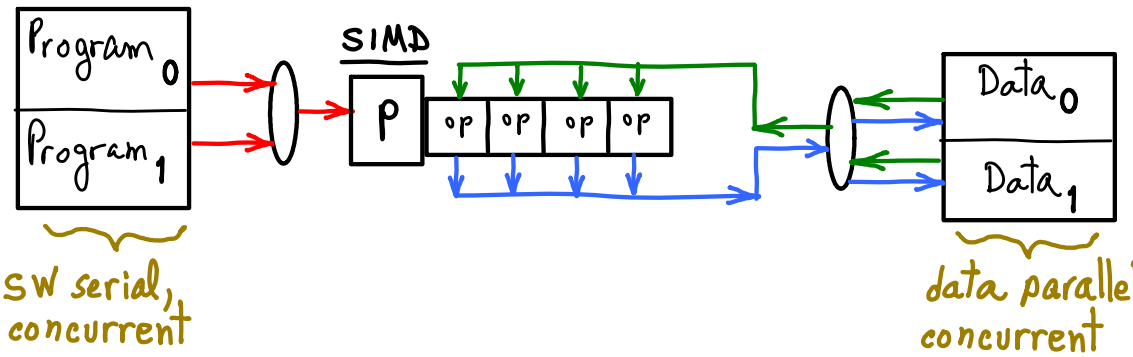
Data serial: single data stream
Data sequential: multi-source



SW serial: single instruction stream
SW sequential: single-process

HW: parallel execution

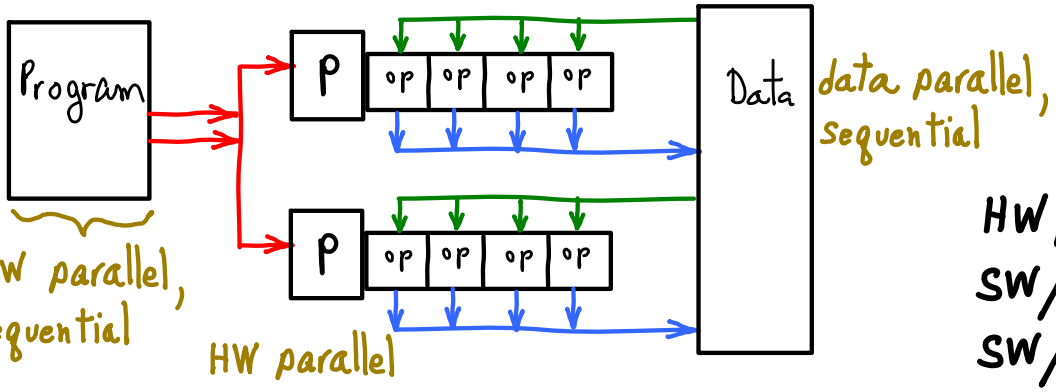
Data parallel: multi-data stream
Data sequential: single-source



SW serial: single stream
SW concurrent: multi-process

HW: parallel

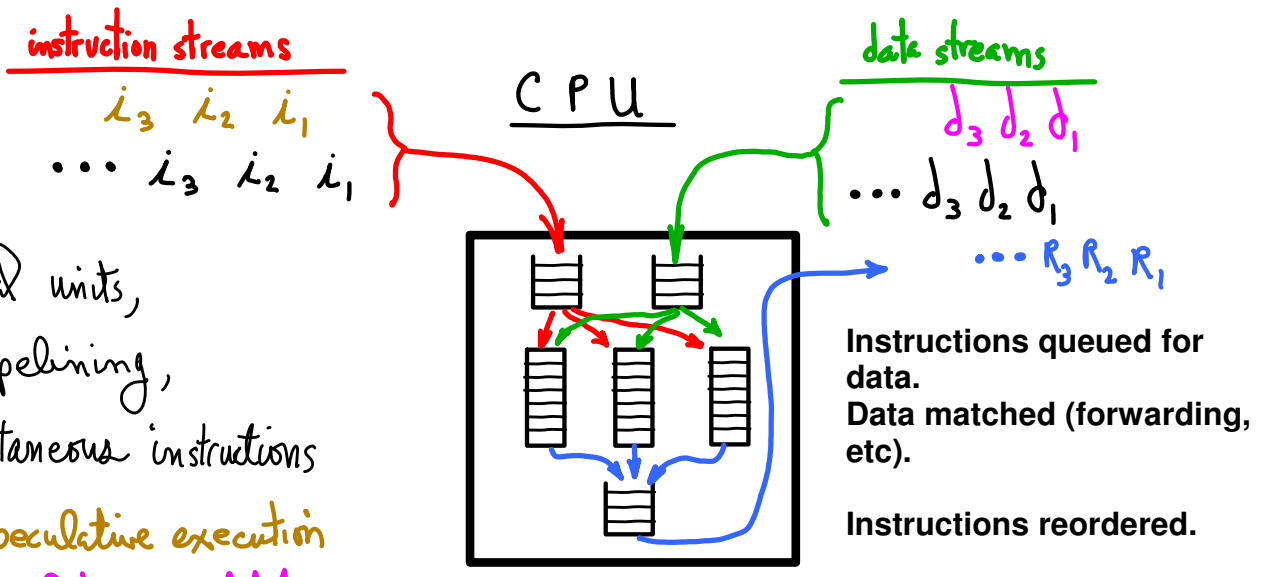
Data parallel: multi-data
Data concurrent: multi-source



etc.

HW parallel: SIMD, MIMD
SW/data: parallel/serial
SW/data: concurrent/seq.

ILP



n functional units,
k-level pipelining,
(n·k) simultaneous instructions

+ speculative execution
+ speculative prefetch

Instructions queued for data.
Data matched (forwarding, etc).

Instructions reordered.

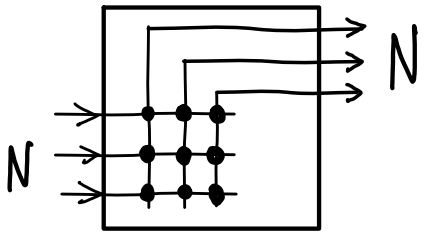
Commits (state change) preserve semantics.

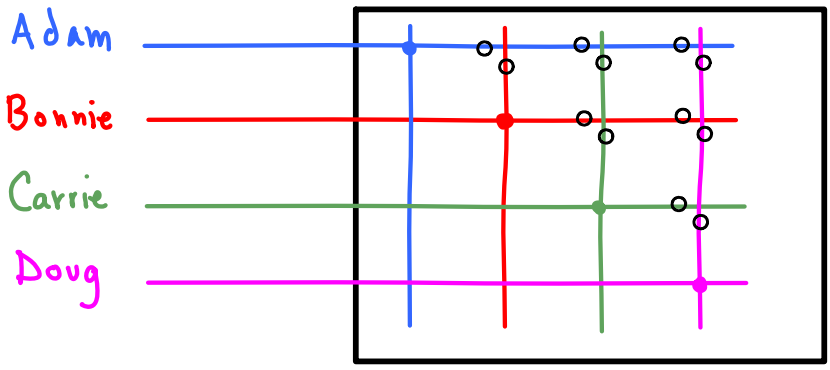
Speculation, nullifying

---

interconnects, switched

N x N
Cross bar
switch



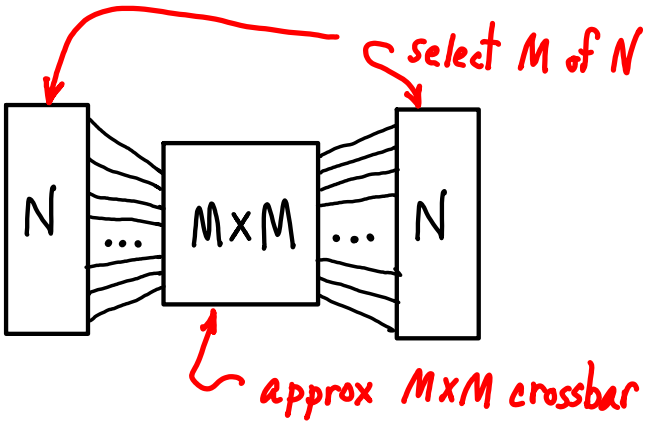


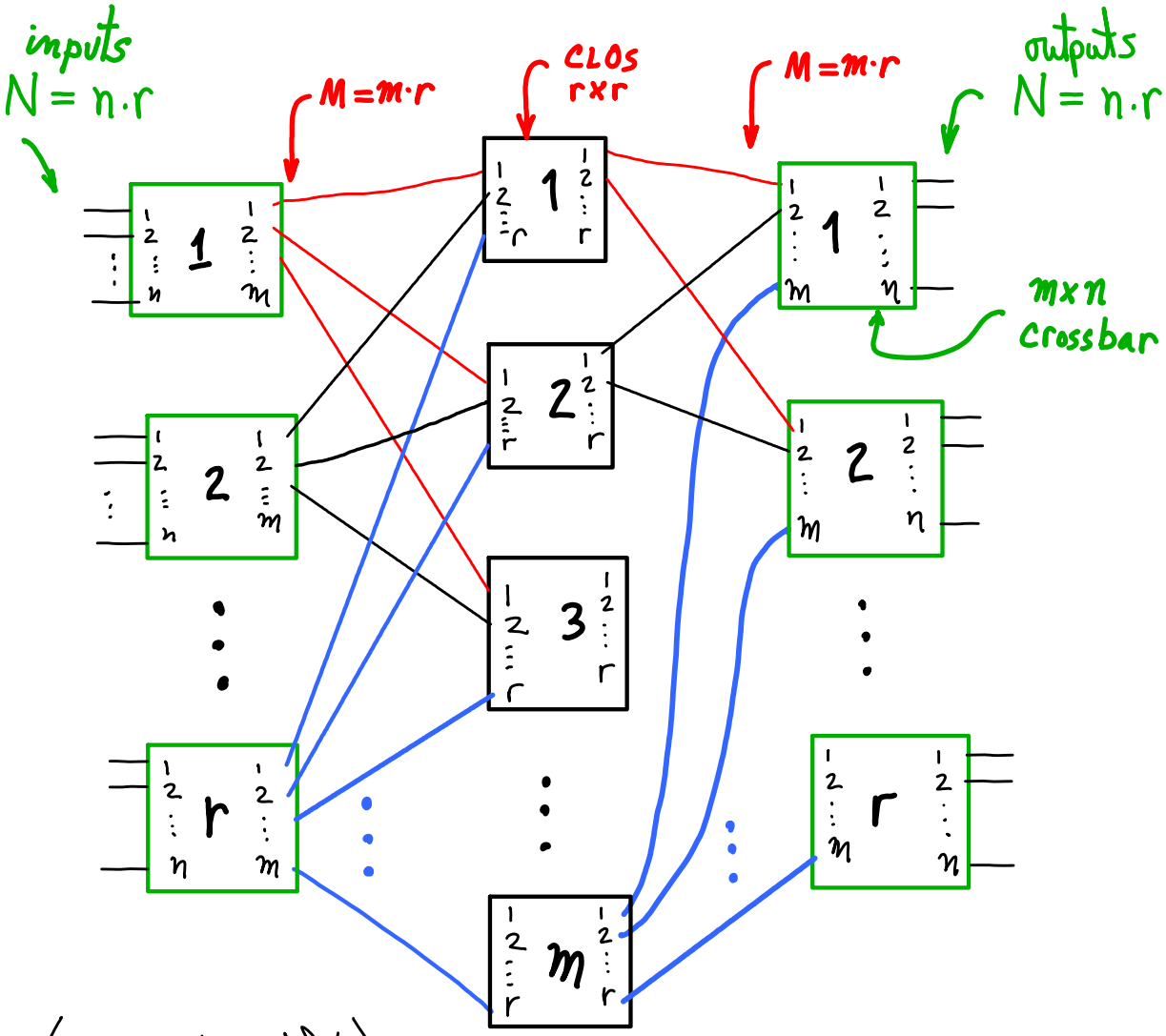
route
- any 1–1 map
- any (N-k) x N * fan-out
- any N x (N-k) * fan-in

* Collisionless (buffering ?)

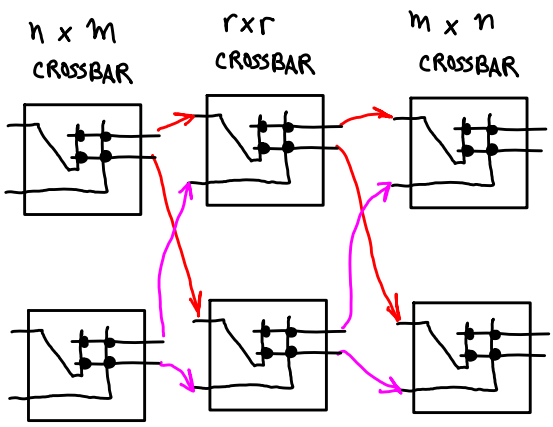
Clos

Approx
N x N

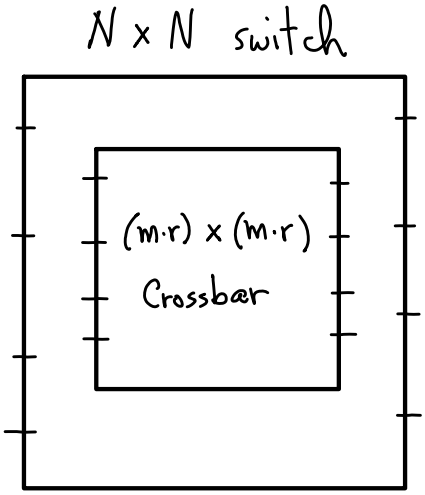




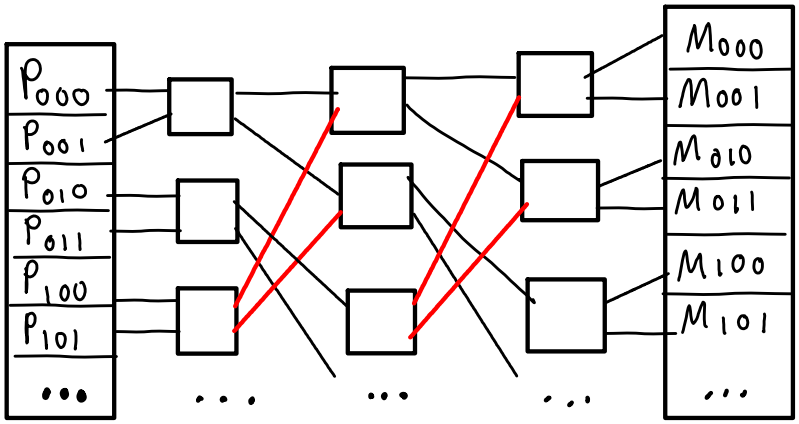
(recursive defn)


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

=


N x N switch
(m·r) x (m·r)
Crossbar

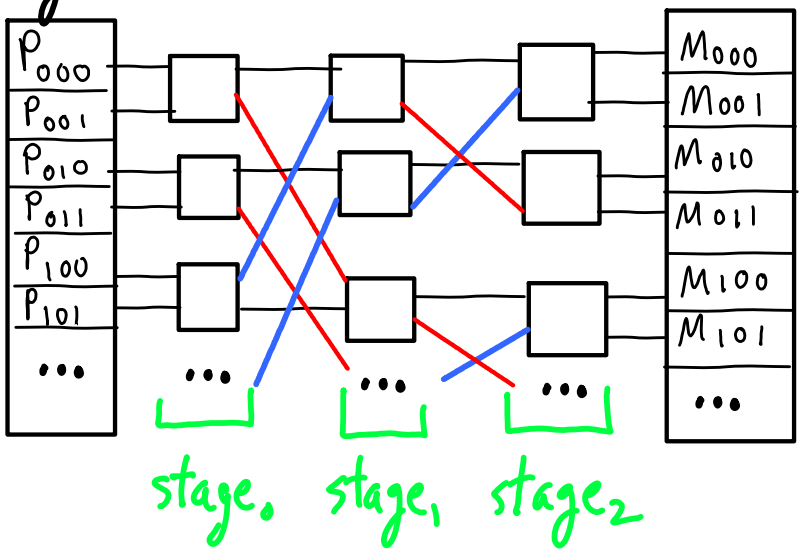
Ω 8×4 interconnect



switch ⇒ blocking

also, can broadcast:

Butterfly 8×4



(Banyan)

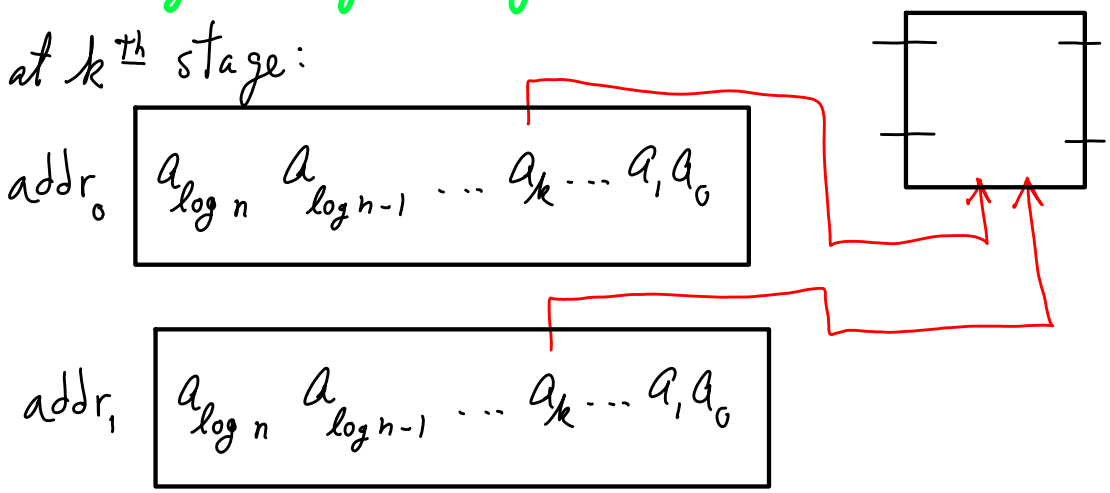
n Ps

n Ms

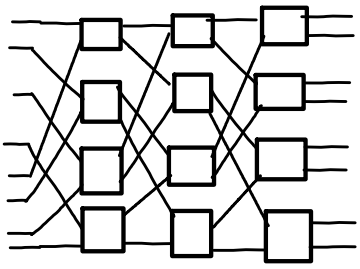
log n stages, @ n/2

log n address bits

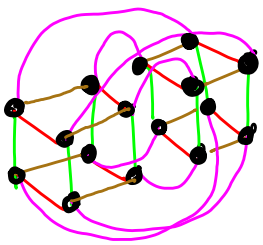
Switch at k^{th} stage:



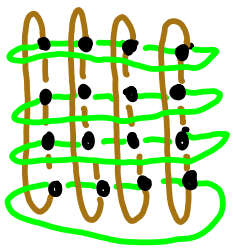
shuffle - exchange (perfect card shuffle)



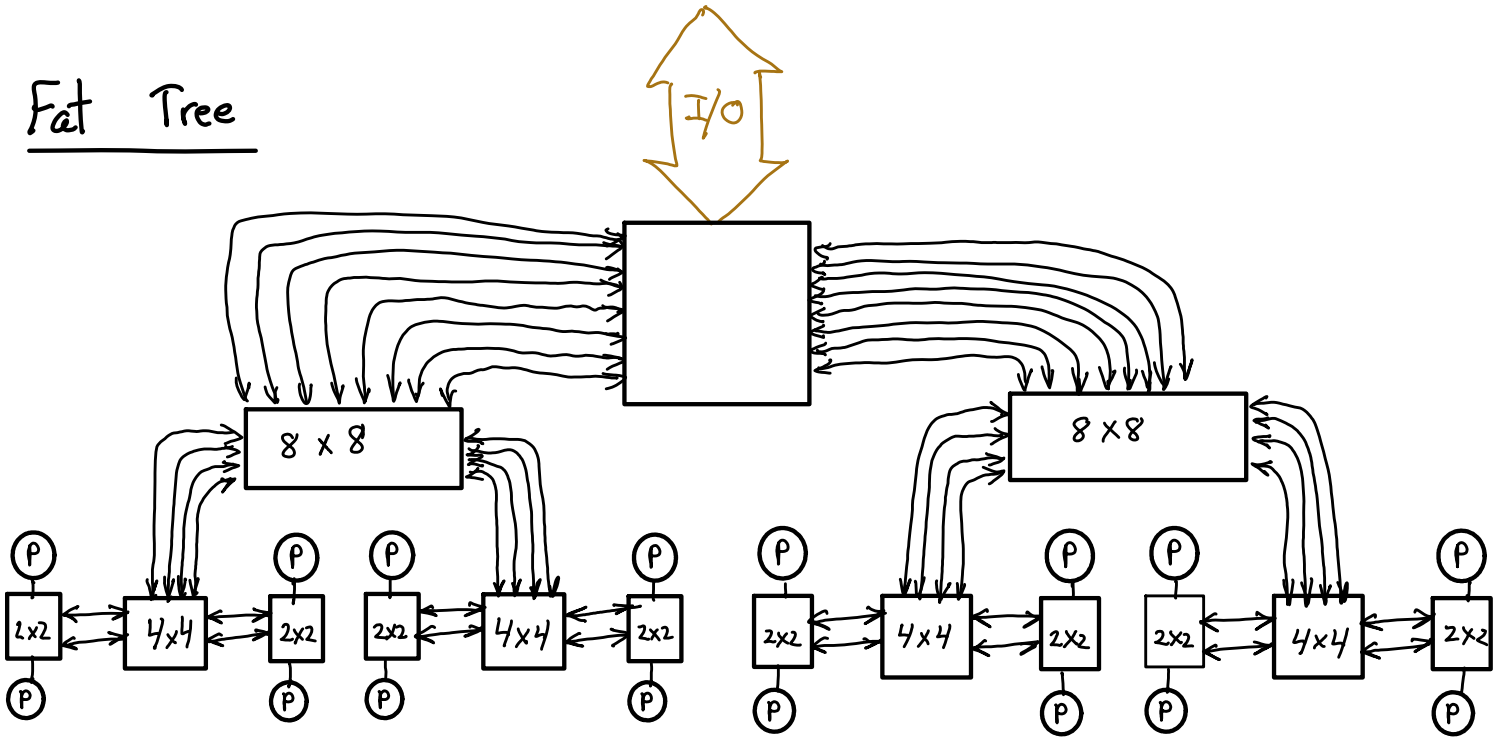
hypercube

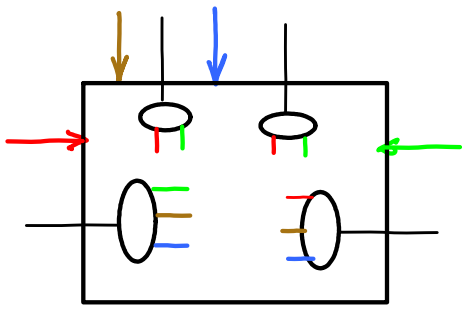


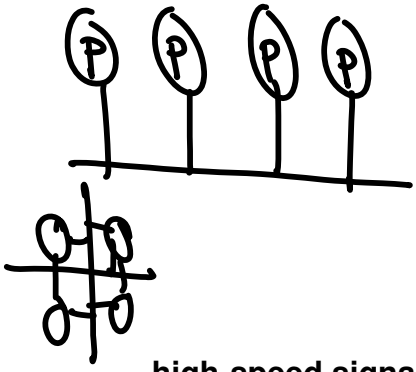
Torus



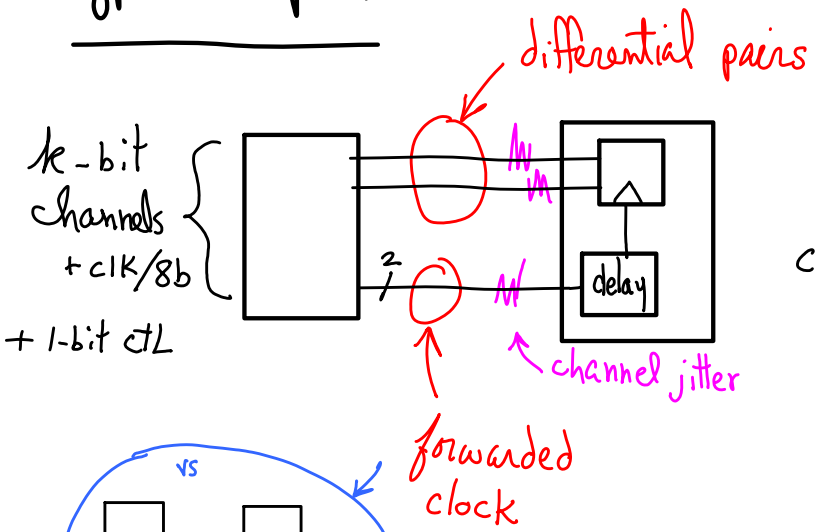
Fat Tree

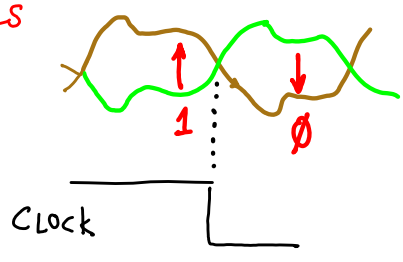




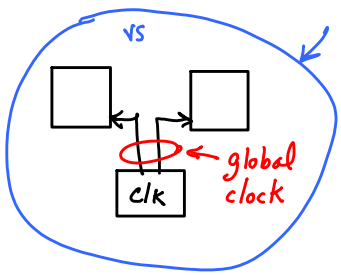


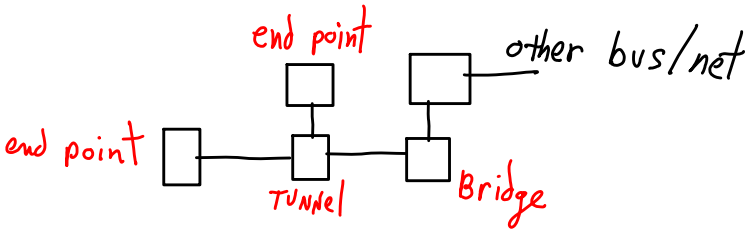
HyperTransport





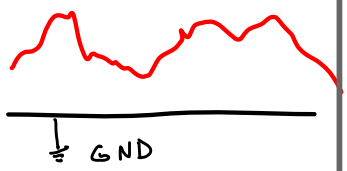
high-speed signals are bouncy (jitter), and corrupted by other signals.





daisy chained, point-to-point

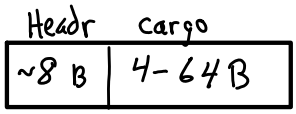
vs.



Detecting whether signal is up or down is difficult.

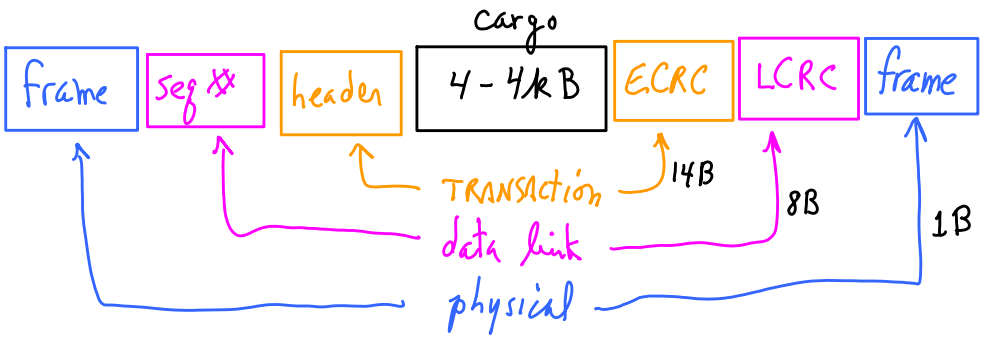
packets { 4-64 B data | 4-8 B CTL }

Packet overhead



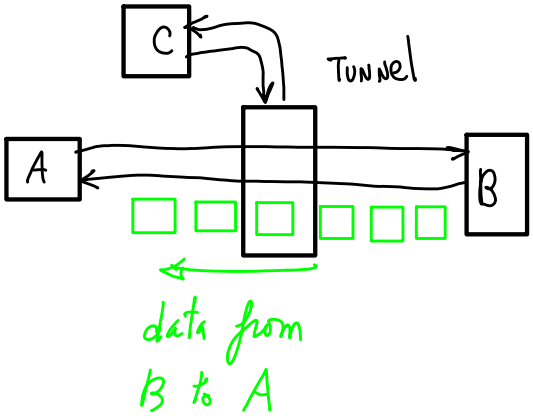
HT

Transaction/data link/physical



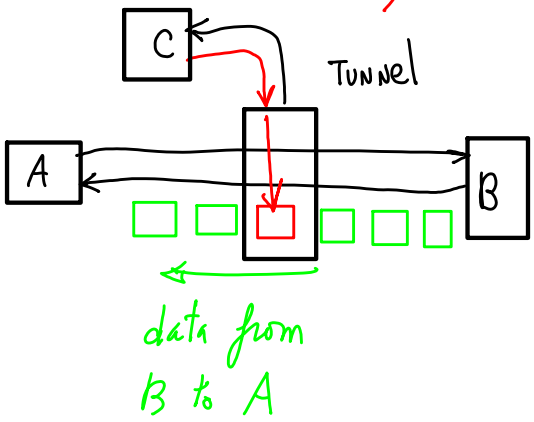
PCIe ~24 B overhead

Priority/interleaving

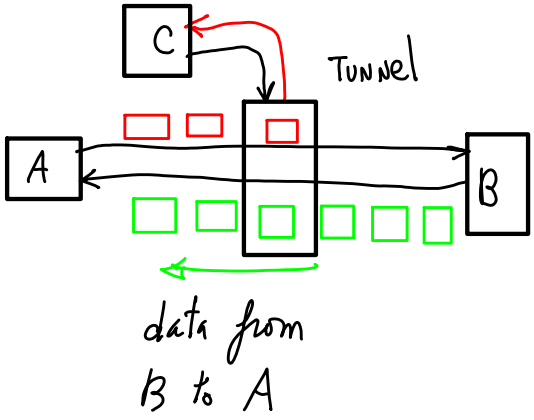


Links in pairs

Priority interleave

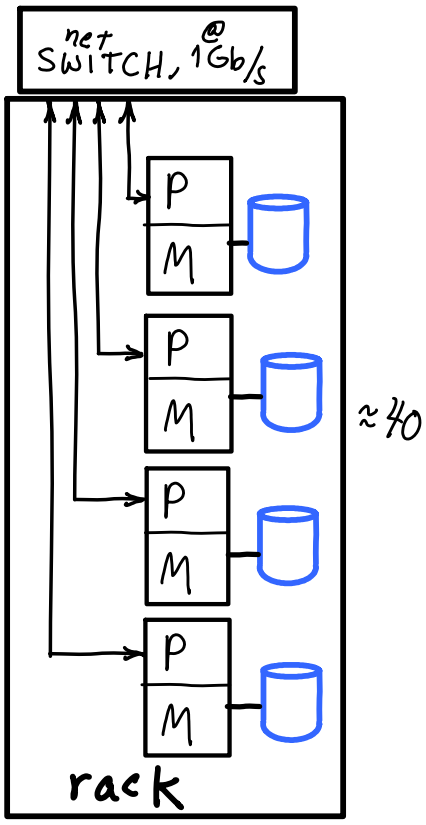


Data from A to C



⇒ Low Latency response

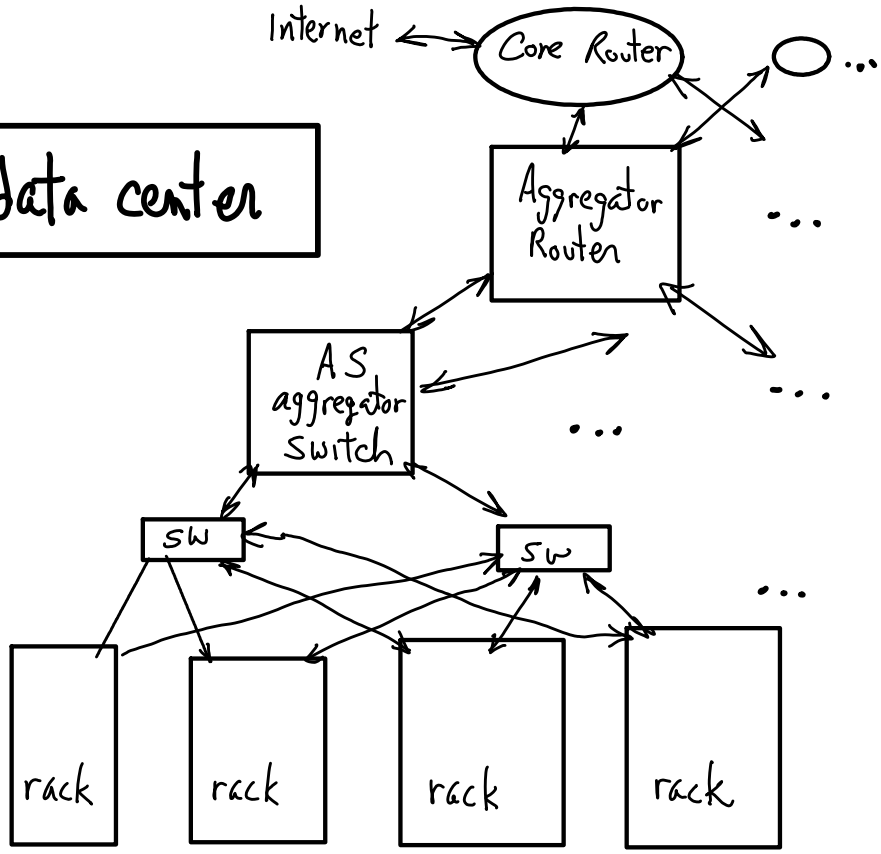
Cluster



non-shared memory, message passing

Thread parallelism
Process parallelism
Task parallelism

data center

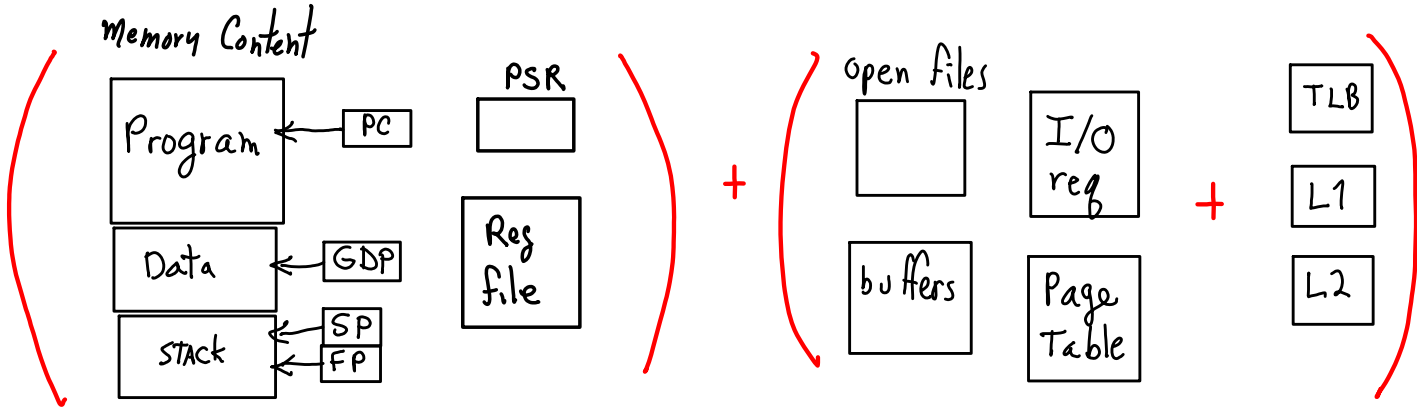


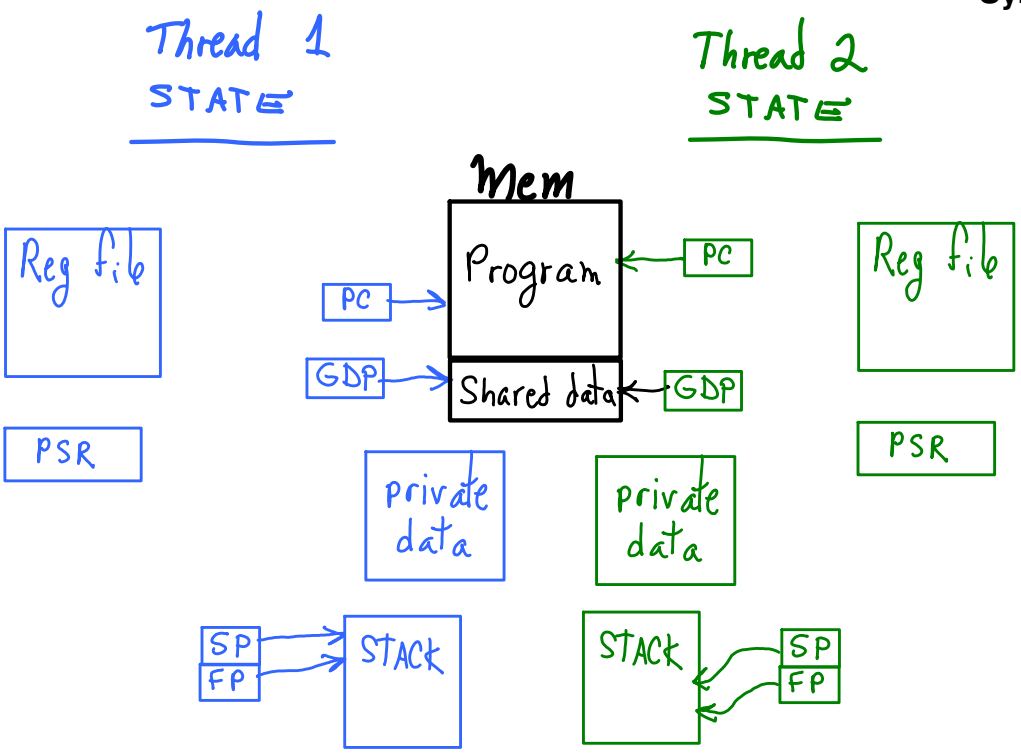
→ difficult algorithm-to-program paradigm
→ slow communication
→ multiple copies of OS, software
→ multiple administration
→ middle ware overhead
→ hot swappable
→ expandable
→ commodity components
→ virtualization, migration, redundancy

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:

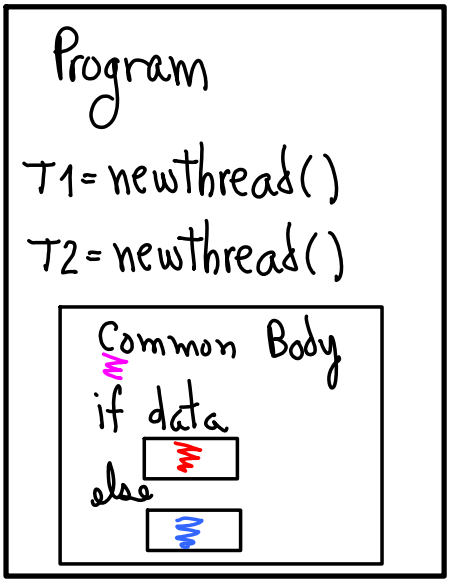
- save state T1
- load state T2

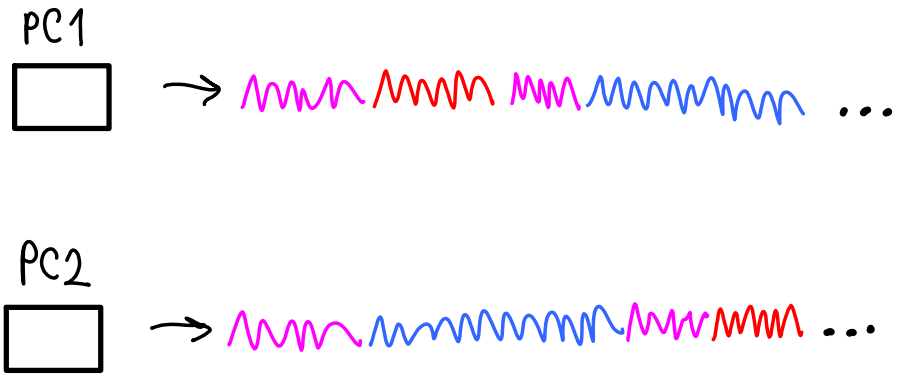
run T2 until

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

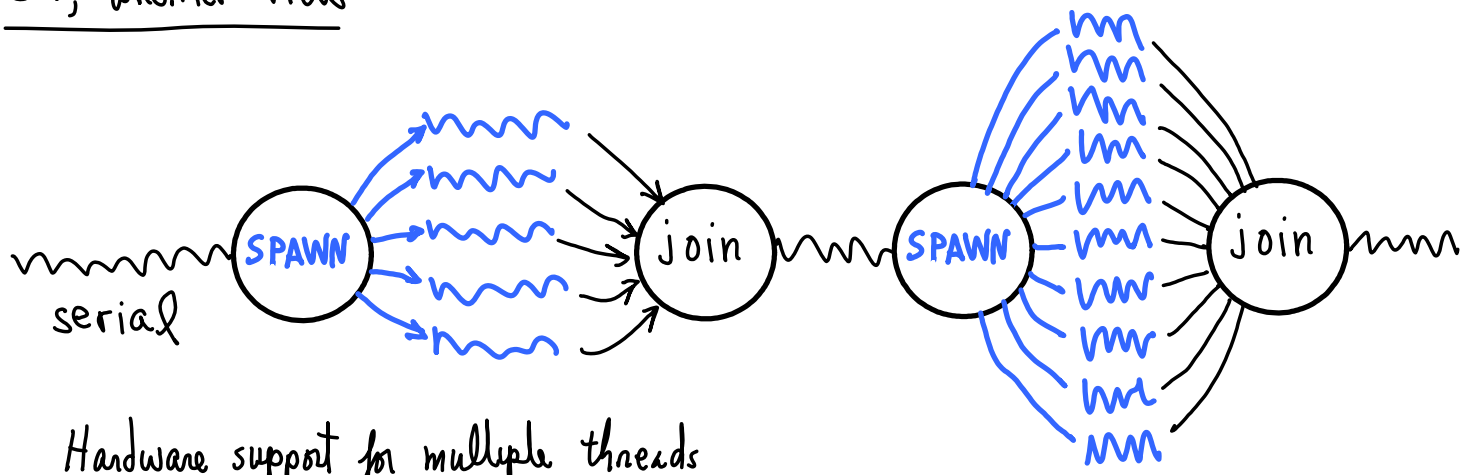
Switch context

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

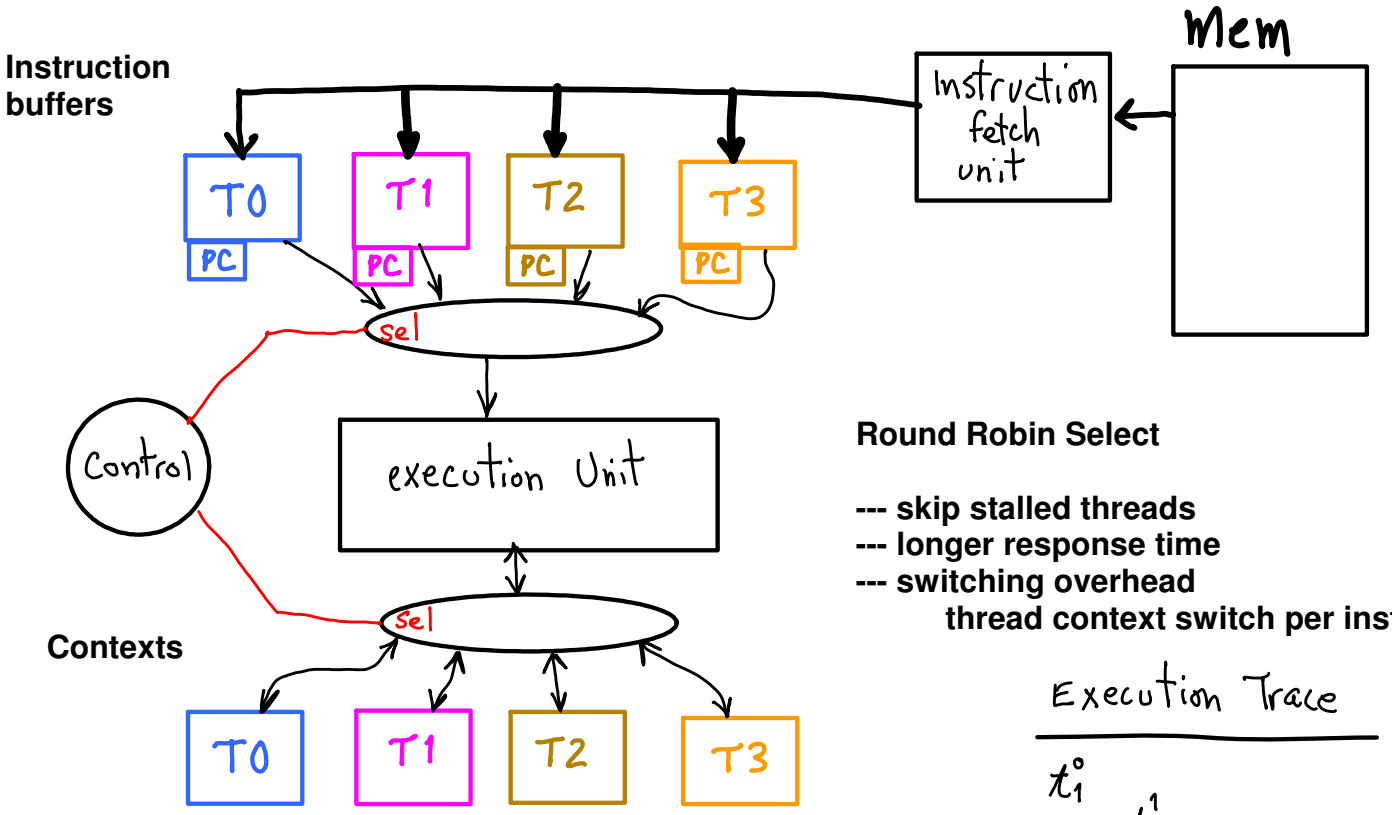
--- TLB content (separate page tables? or shared?)
--- hardware switch: thread ID labeled (TID)

--- MULTIPLE THREADS from MULTIPLE PROCESSES: PID + TID

--- Larger state to consider (page tables, file and IO tables and buffers)
--- Page Tables
--- File and IO tables and buffers
--- TLB and Cache content

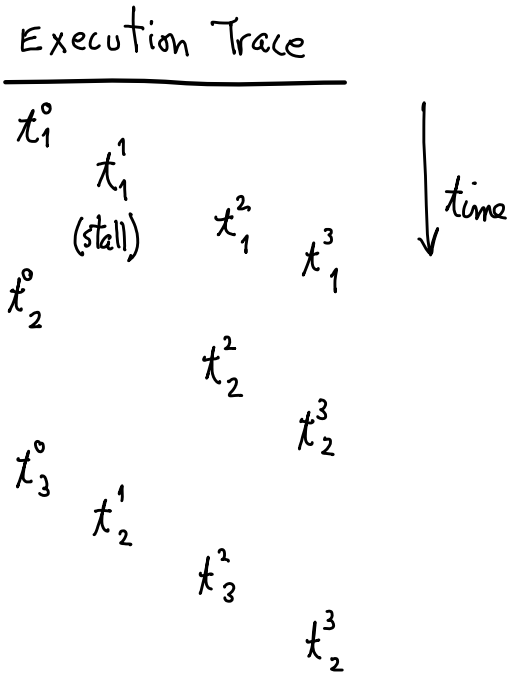
OS policy not known to HW designer?

Fine-grained multi-threading

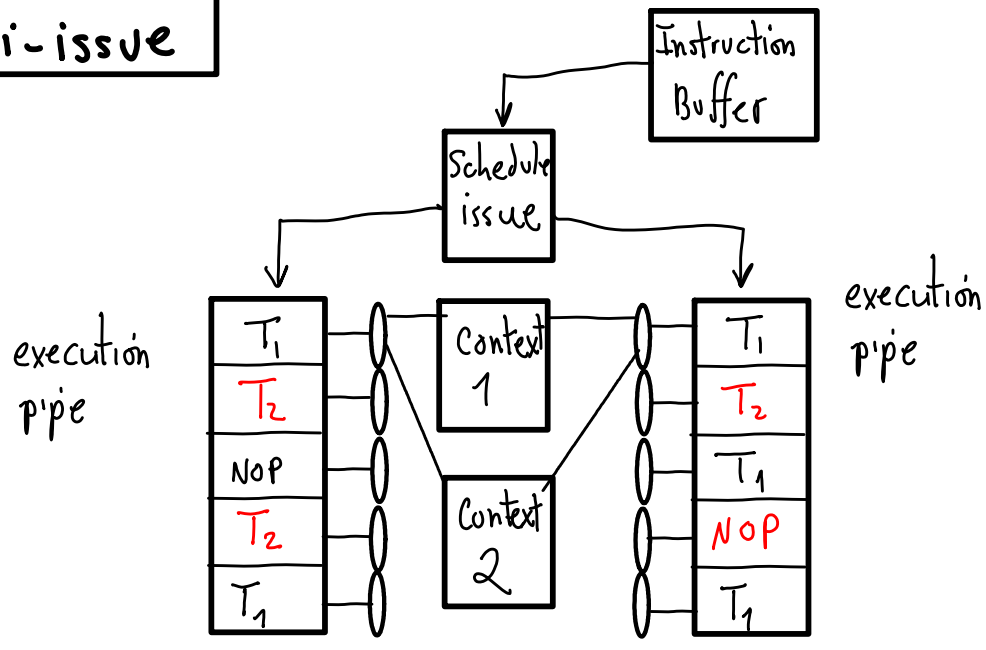


Round Robin Select

--- skip stalled threads
--- longer response time
--- switching overhead
thread context switch per instruction



multi-issue



Thread context switch per stage.

Coarse Grained

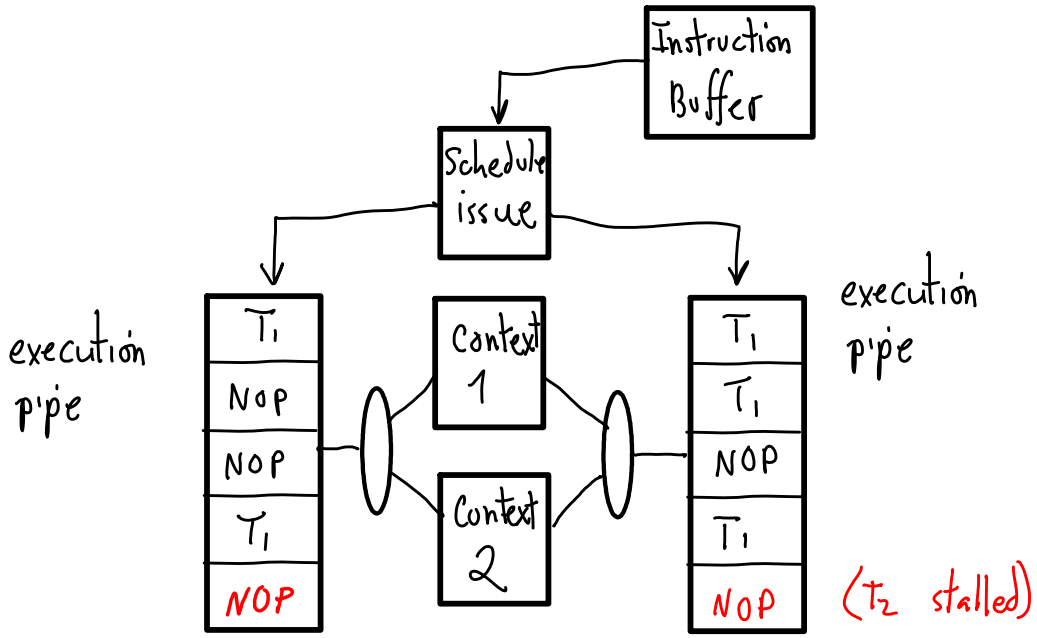
Same HW, but schedule same thread until stalled.

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

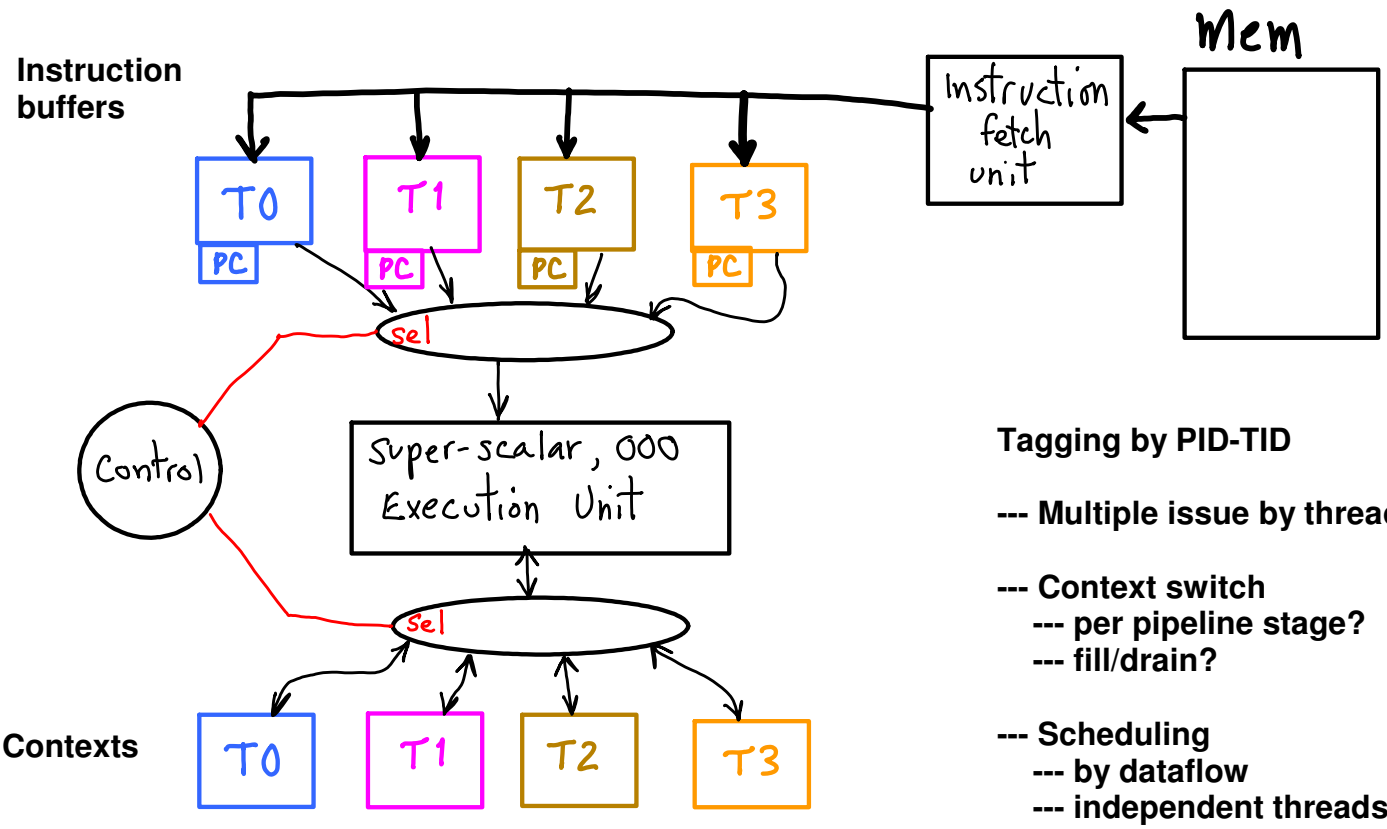
--- Faster response time
--- lower switching overhead
    --- affords more complex control
    --- faster execution between
--- Starvation now a problem?

Multi-issue



Context switched per pipe.

(Simultaneus)(Hyper) Threading

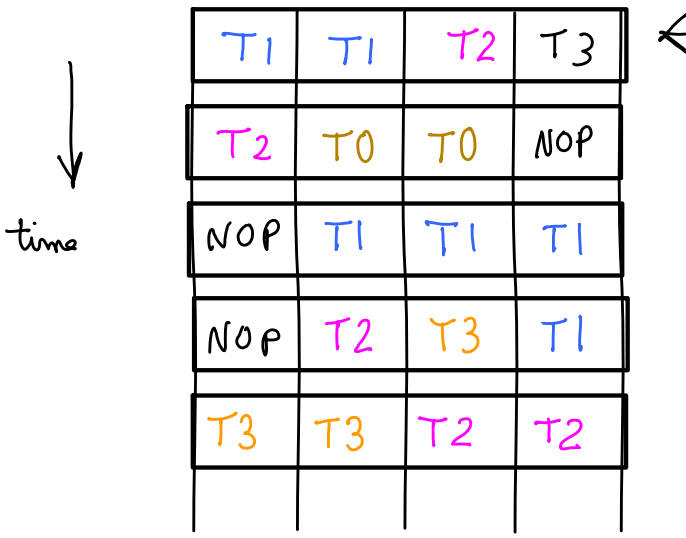


Tagging by PID-TID

--- Multiple issue by thread

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

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



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

Single threaded execution

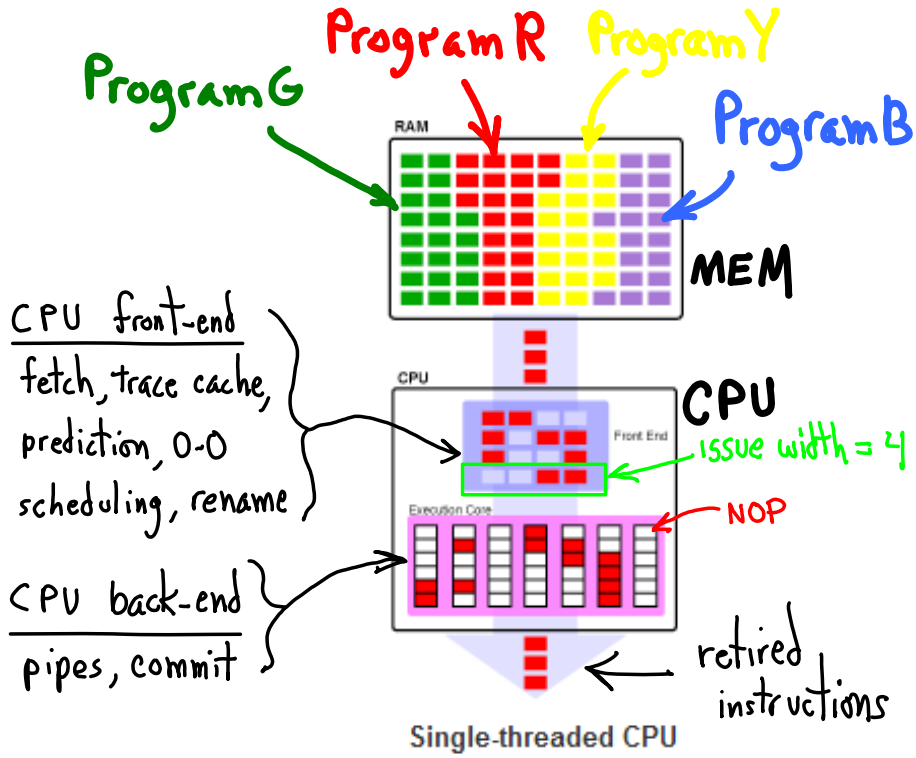
Multi-tasking

---- 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

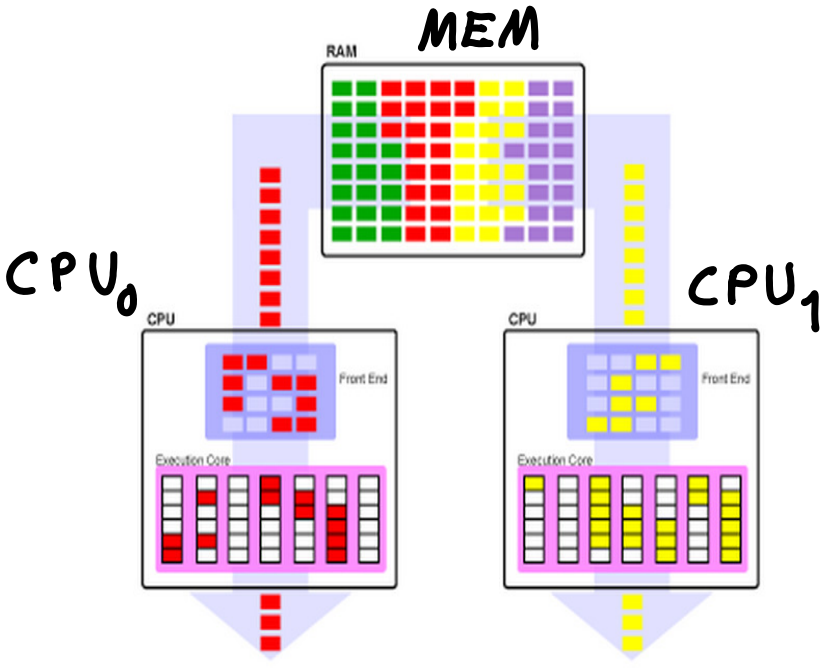
Credits:
Introduction to Multithreading, Superthreading and Hyperthreading
By Jon Stokes

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, ...

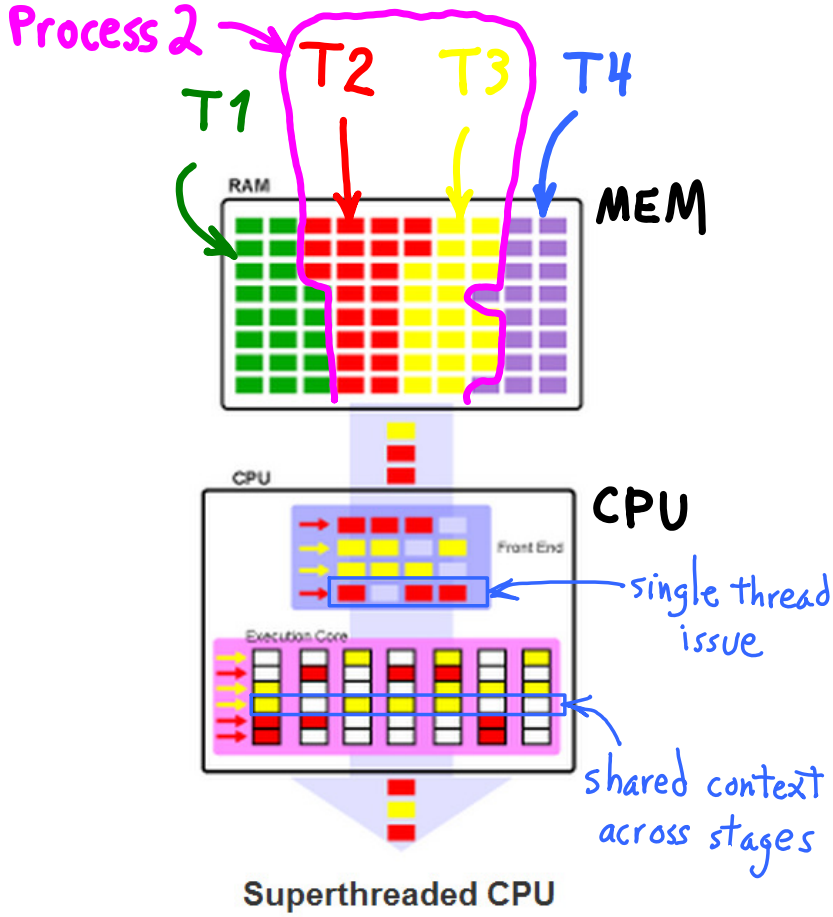
---- Single process, multiple thread execution

---- 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

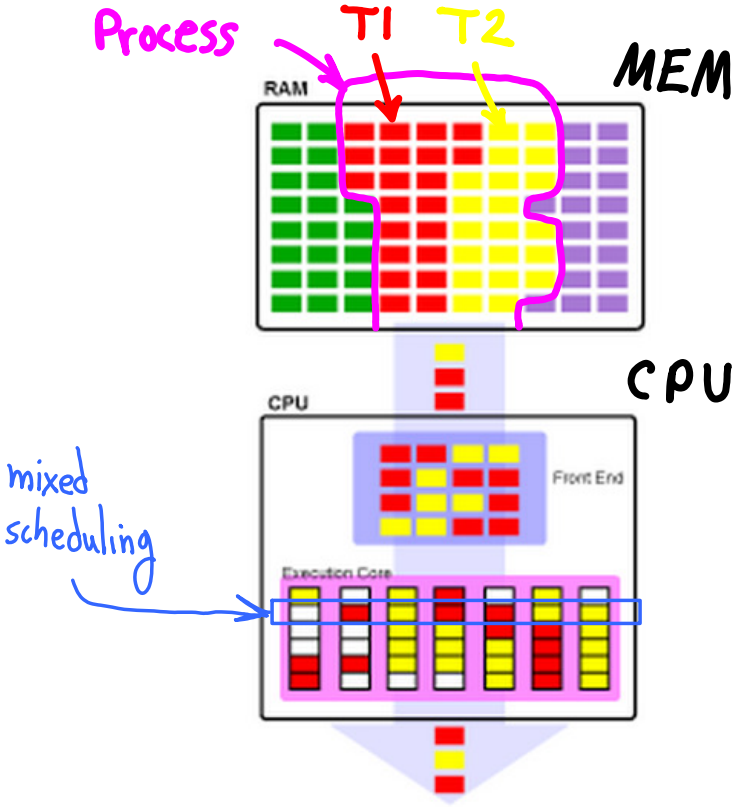
---- Concurrent Processes
--- context switching

---- 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



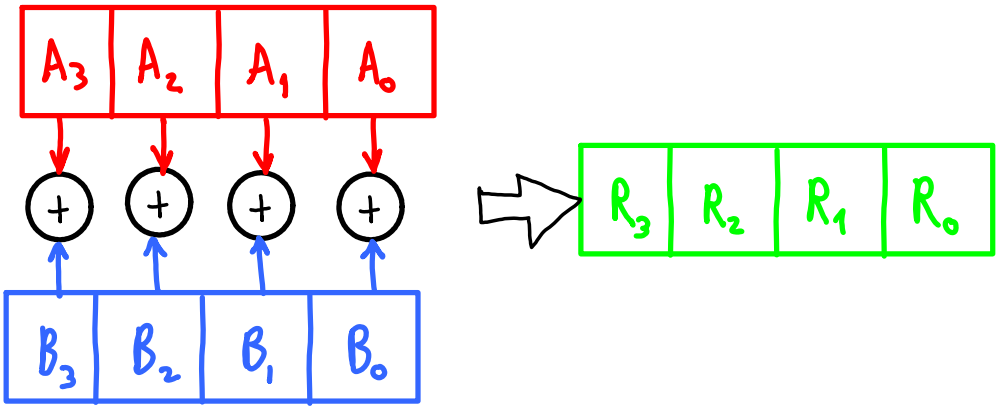
Hyper-threaded CPU

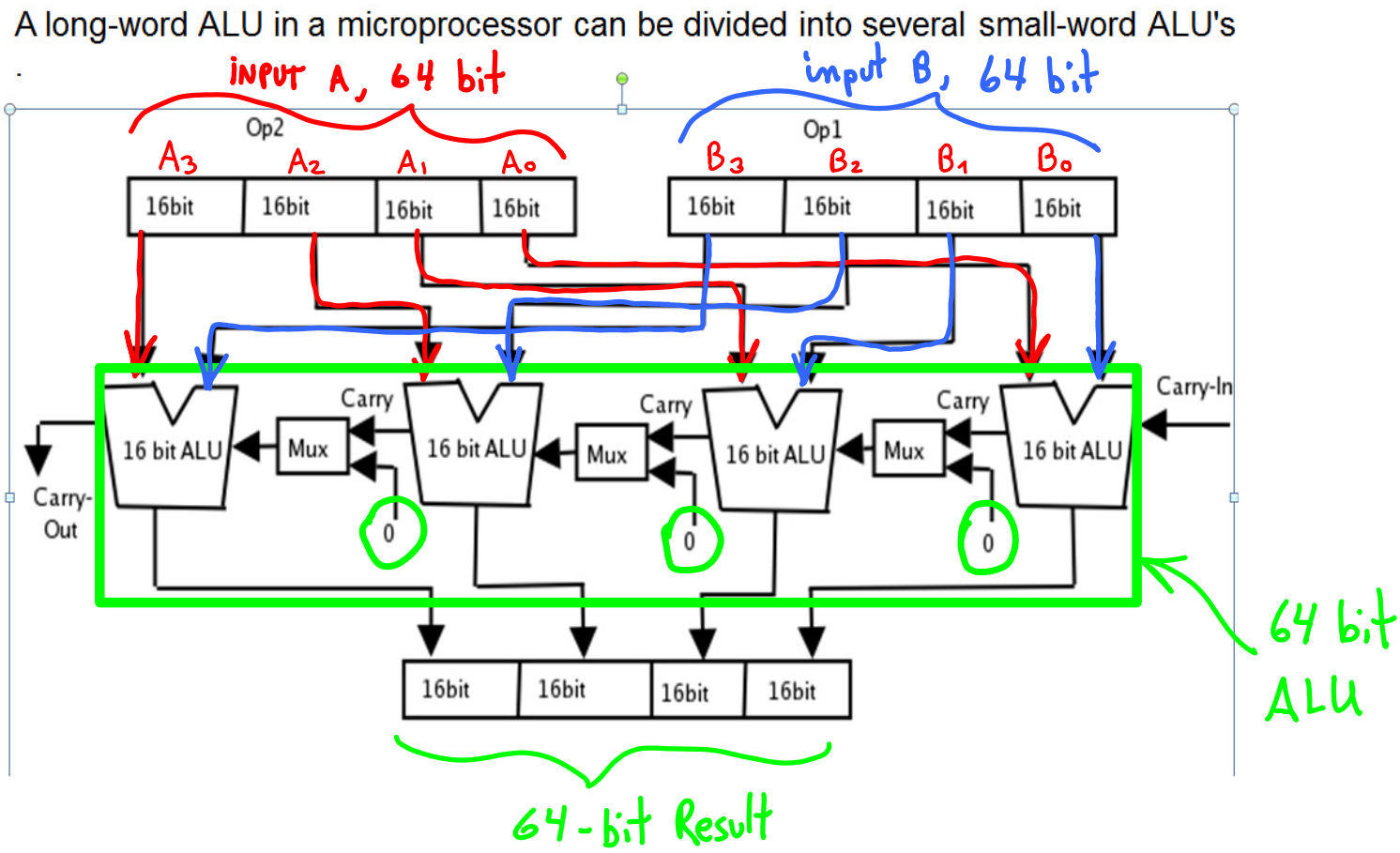
Modification of existing O-O-O CPU
⇒ 10% added cost, $s^{th} > 2$

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

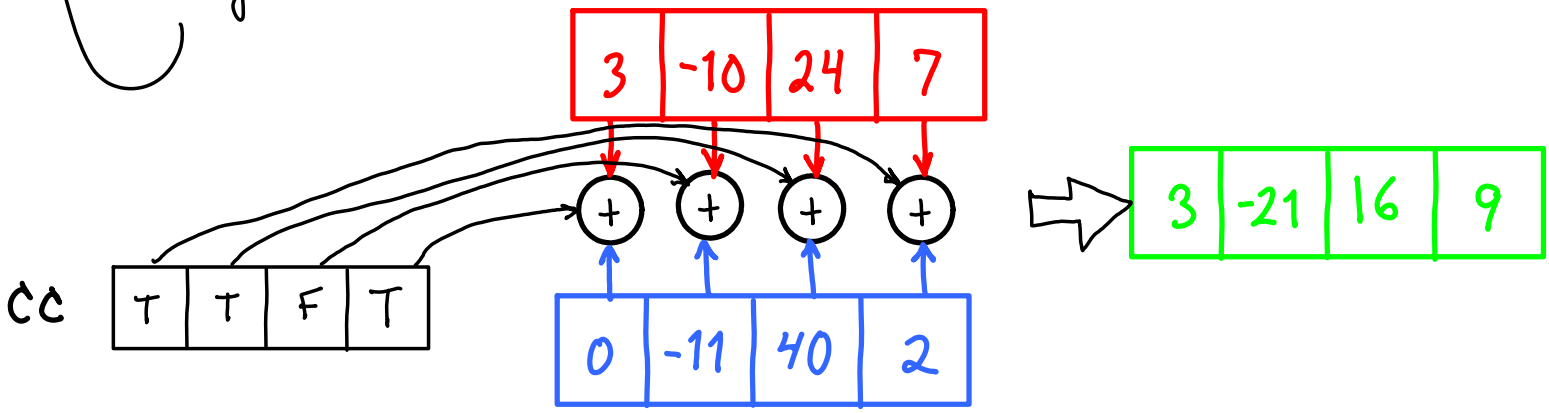
$R_i \leftarrow (-A_i) + B_i$

endif

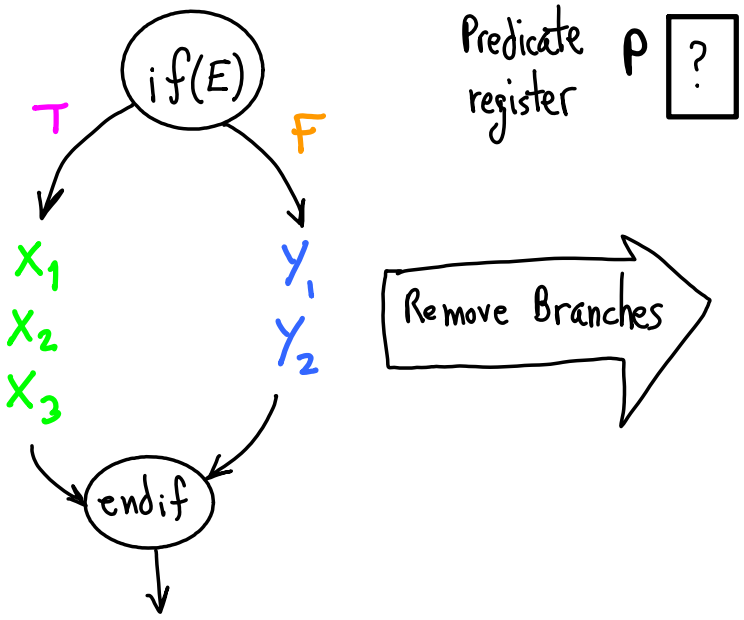
$\Rightarrow$ $(A-B) \Rightarrow CC$ (predicate vector)

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	SRC0	SRC1	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