Constructive Computer Architecture

Introduction to (simultaneous) multithreading

How to go faster?

- Lost opportunities, sometimes the processor is mostly dead:
  1. On load miss
  2. When stalling for dependencies
  3. On branch misprediction

Previously in 6.1920

- Processor does 2 insts per cycle
- Cost ☹:
  - Increased complexity in control logic (hard to debug/verify)
  - Increased area/energy if unused:
    - More ports in RF …
- Noncost ☺:
  - No changes to the programs!
  - Size of cache/size of Rf

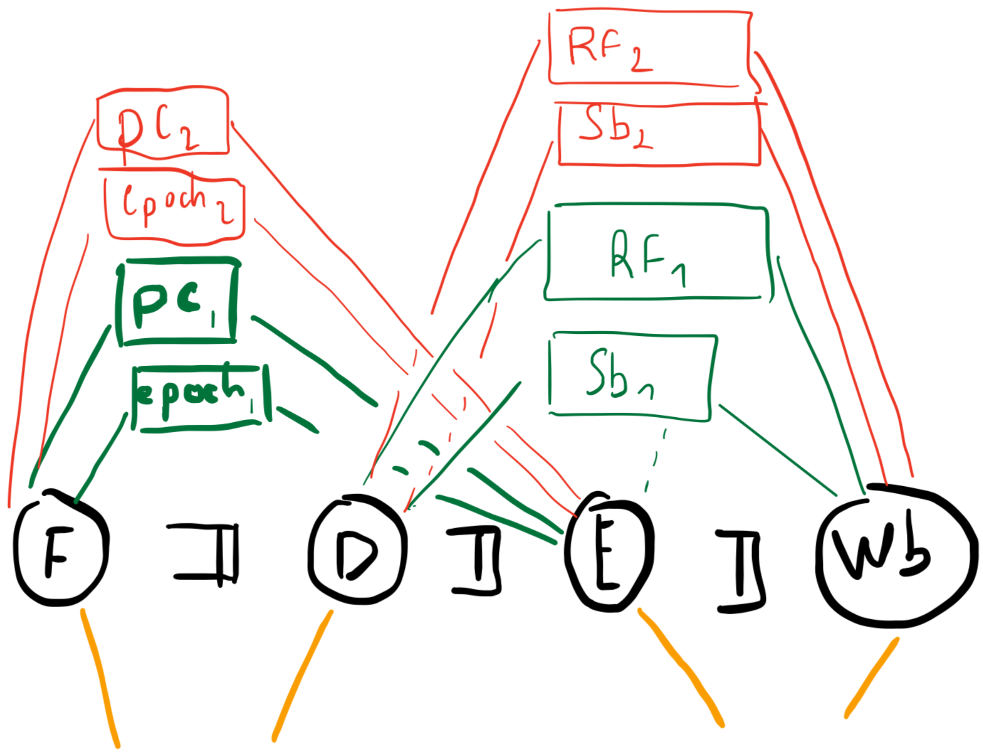
Yet another idea

- So far, processor runs 1 program
- Why not making it run 2 (or k) programs?
  - pc_x, rf_x[_]
  - pc_y, rf_y[_]

Multithreading

- Run one instruction of each program each cycle?
  - Not necessarily, almost never in the rest of this lecture

Preliminary diagram



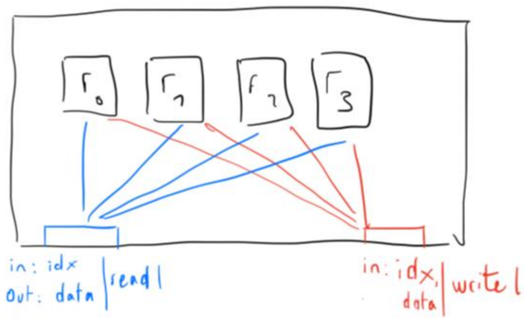
Multithreading (MT)

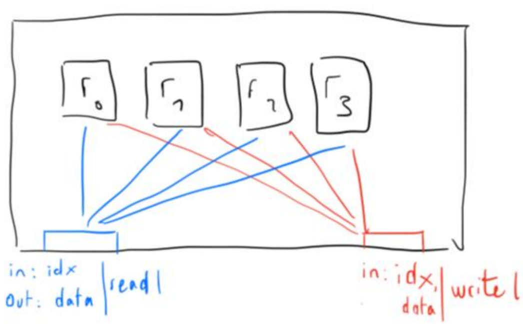
- Noncost ☺:
  - Lower complexity in control logic
- Cost ☹:
  - Changes to the programs!
  - Increases number of Rf
  - Potentially requires bigger caches

Tradeoffs in the RF

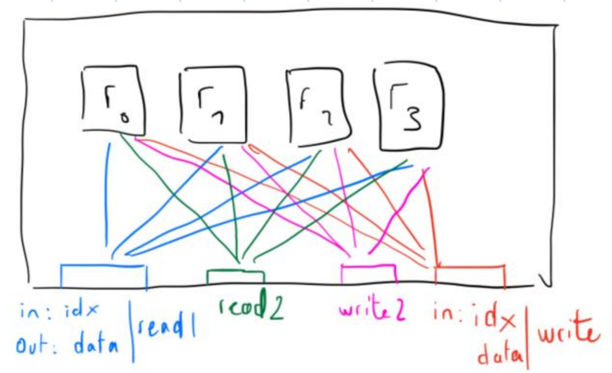
- RF is on the cost side <u>for different reasons</u>.

RF for multithreading





Tuesday's RF for superscalar



Tradeoffs in the RF

- Relative costs of
  - N registers of size B, R read ports, W write ports, rfcost(N,B,R,W)?
  - rfcost is (critical path, area, energy)
- Increasing the number of ports is typically quite expensive

Wait – Back to superscalar's lecture

Side-notes – Superscalar throwback

- Split the 32 registers in 2 subRF of 16 registers
  - Each subRF has 2 read ports, 1 write port
  - Take 1 cycle if the two instructions don't collide
  - Take 2 cycles if collision
- Ask the compiler to be nice and generate collision-free instructions

Return to today's lecture

Refining the idea

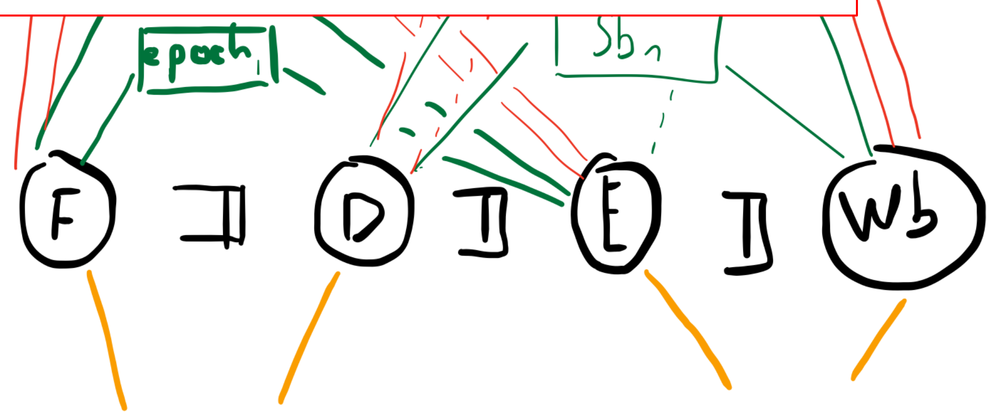
- Running 2 programs? Ok, but how?
  1. “Fine-grain multithreading”
  2. “Coarse-grain multithreading”
  3. “Simultaneous multithreading”

Let’s discuss at a high-level first, then details

1.Fine-grained MT

- Cycle 0: Fetch from pc1
- Cycle 1: Fetch from pc2

Reduce stalling due to dependencies between registers!



2.Coarse-grained MT

- Cycle 0: Fetch from pc1
- Cycle 1: Fetch from pc1

ss

Does not help with
dependencies between registers.
Only target load misses!

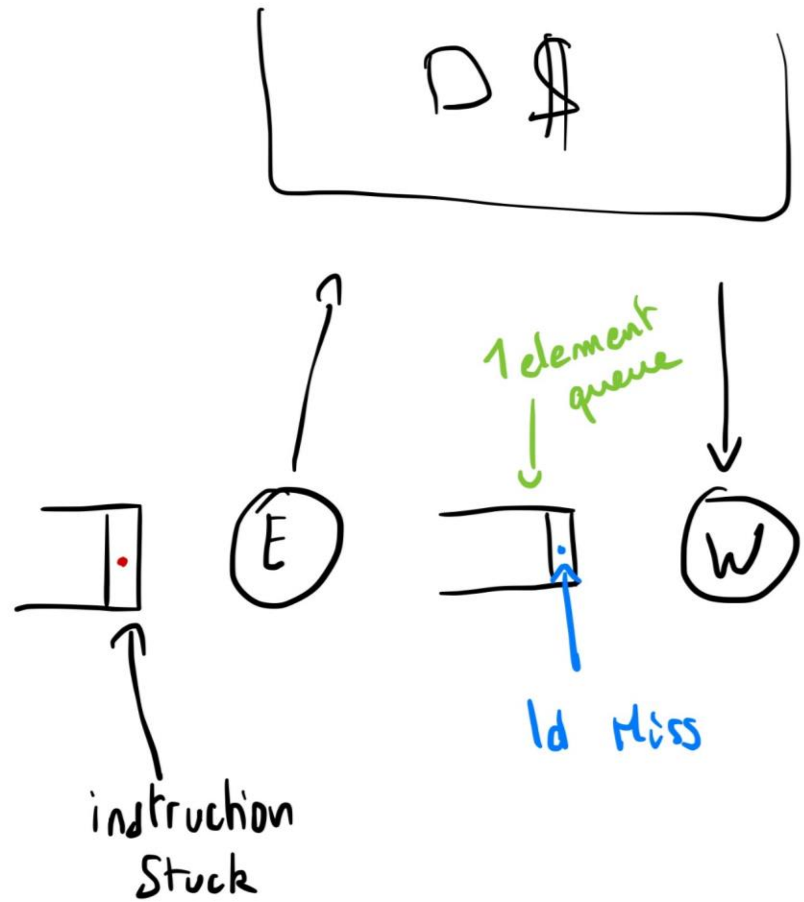
3. SMT (Simultaneous)

- Fetch instructions from both threads simultaneously ("combined with superscalar")

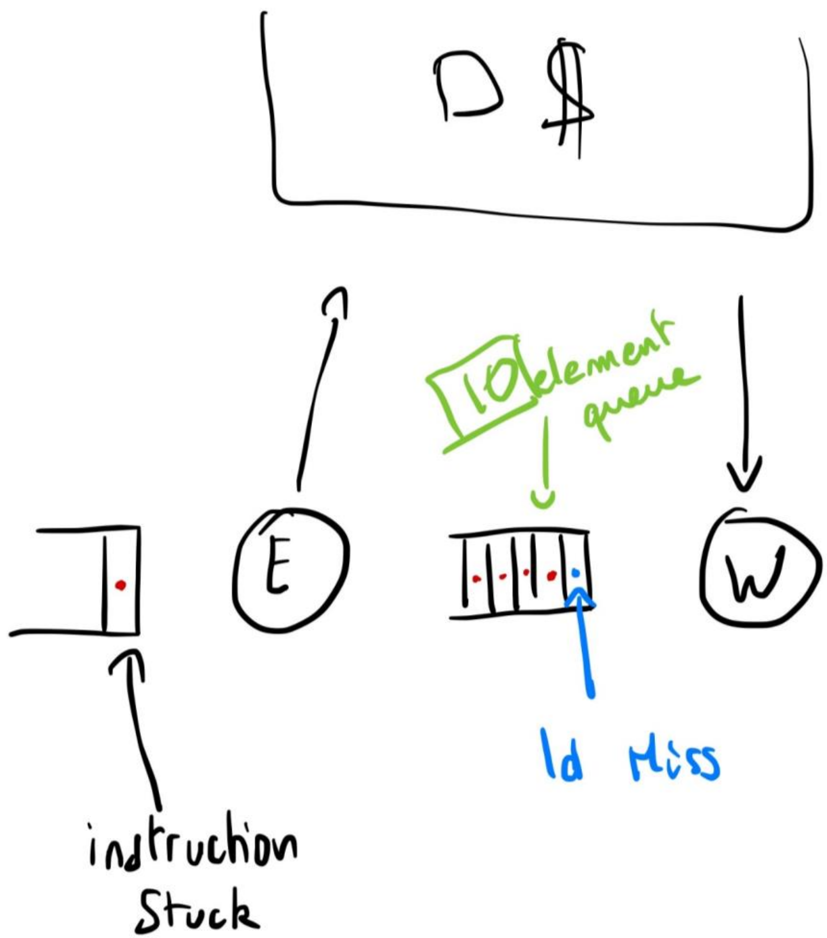
Great, does it work?

Yes (part one of lab) but No (part two of lab)

- Functionally yes.
- It does not wo backpressure!



Not a solution



We are stuck behind the blue truck

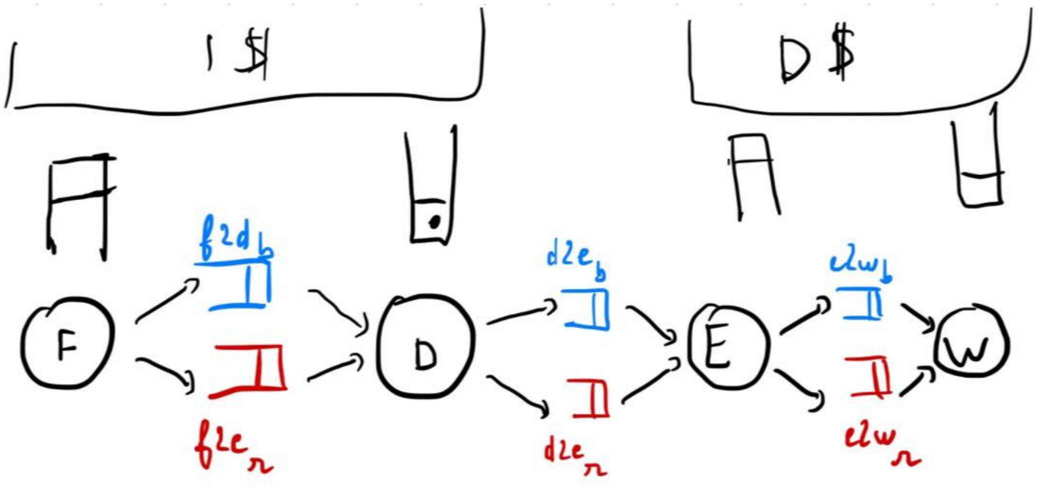
Latency Load miss ~40/60 cycles – gigantic structure

Solution

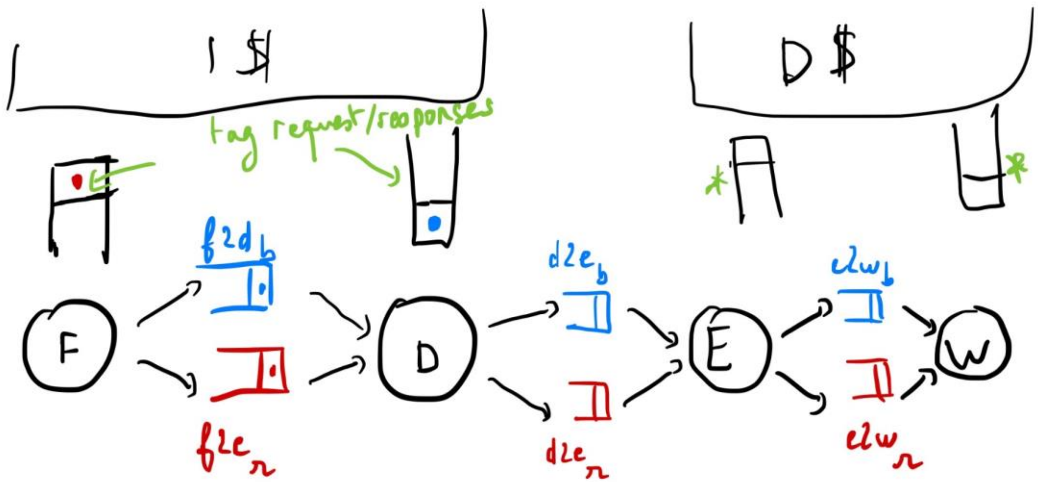
- Use two 1-element queues for e2w, one for each thread

- Is it "necessary" to also use two queues for f2d and d2e?
  1. *In any case, this is only about performance*
  2. For perf, it depends: no for Coarse-grain, yes for the others.

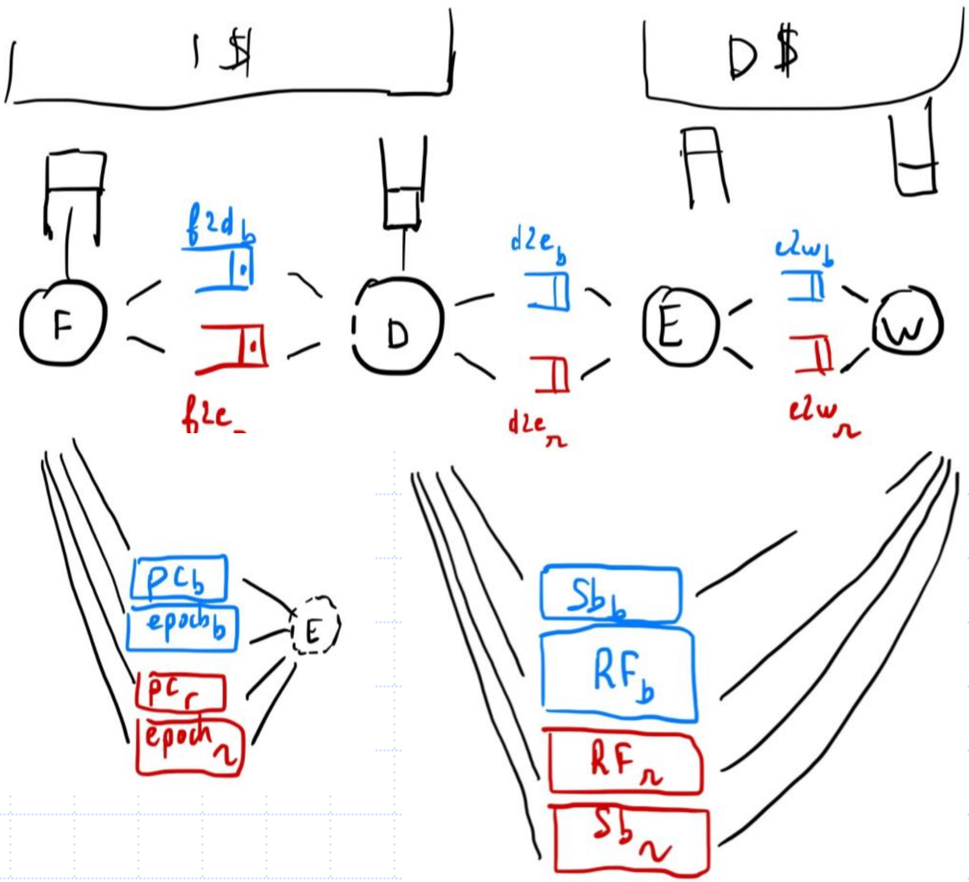
More accurate diagram



Morer accurate diagram



Drawing more



1. Are there potential choices for scheduling?

2. Wait, did we just duplicate everything?

Round-robin scheduling

- High-level:
  - Try to be fair and run both threads
    - It is good to avoid dependencies!
    - “Don't put all your eggs in one basket”
  - If one thread is stuck, make progress on the other one

Round-robin in Execute

```
// Original code:
rule execute if (!starting);
    d2e.deq();
    let current_inst = d2e.first();
    if (current_inst.epoch == epoch[0]) begin
        do the current_inst
        if ppc != nextpc(current_inst), redirect and toggle
epoch
    end else begin
       squash current_inst
    end
endrule
```

Round-robin in Execute

```
rule execute if (!starting);
    let current_instb = d2e_b.first();
    let current_instr = d2e_r.first();
    // choose which one to run, dequeue from that queue
    …
endrule
```

Does not work, execute gets stuck
if I don’t have both red and blue insts available

Round-robin in Execute

```
Reg#(Bool) priority <- mkReg(True);
rule execute if (!starting);
    if (priority) begin
        if (d2e_b.notEmpty) begin
            d2e_b.deq();
            let current_inst = d2e_b.first();
            Execute current_ins and push in e2w_b
        end else begin
            d2e_r.deq();
            let current_inst = d2e_r.first();
            Execute current_ins and push in e2w_r
        end
   end else begin
        // […]
   end
   priority <= !priority;
endrule
```

Round-robin in Execute

```
// The  ellipsis: […]
else begin
        if (d2e_r.notEmpty) begin
            d2e_r.deq();
            let current_inst = d2e_r.first();
            Execute current_ins and push in e2w_r
        end else begin
            d2e_b.deq();
            let current_inst = d2e_b.first();
            Execute current_ins and push in e2w_b
        end
   end
   priority <= !priority;
endrule
```

Handle control instructions

- If instruction comes from
- How to correct from misprediction?
  - Two epochs: one per thread
  - Execute rule use the appropriate epoch (depending on the source d2e_b or d2e_r) to filter out the instruction
  - Modify the appropriate pc

Exploiting choices

https://dl.acm.org/doi/pdf/10.1145/232973.232993

- Every cycle, which thread should we fetch from? Execute from? Etc...
- There is a *scheduling* choice:
  - Greedy (1 except if one stuck, then 2)
  - Alternate?
  - Random?
  - Anything smarter?

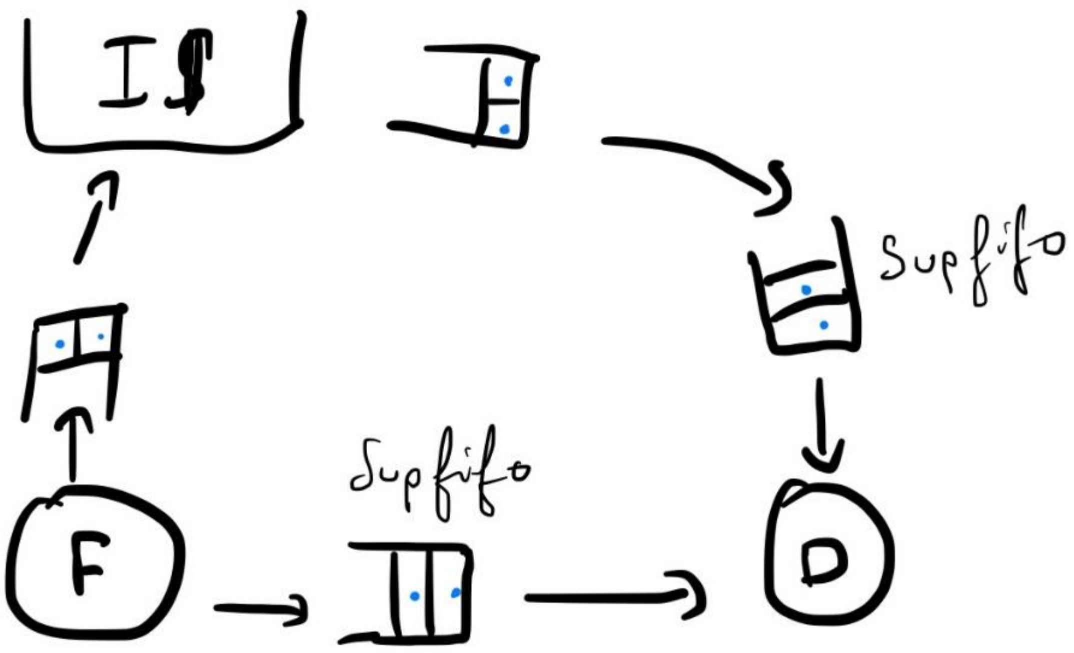
More choices

- Interesting policies:
  - #in-flight branches on each thread. Pick thread with minimal number of branches (*if possible)* **Why is it good?**
  - #in-flight misses on each thread
  - #in-flight instructions (priority to minimal in-flight instructions)
- Those ideas even more important when we have a superscalar machine

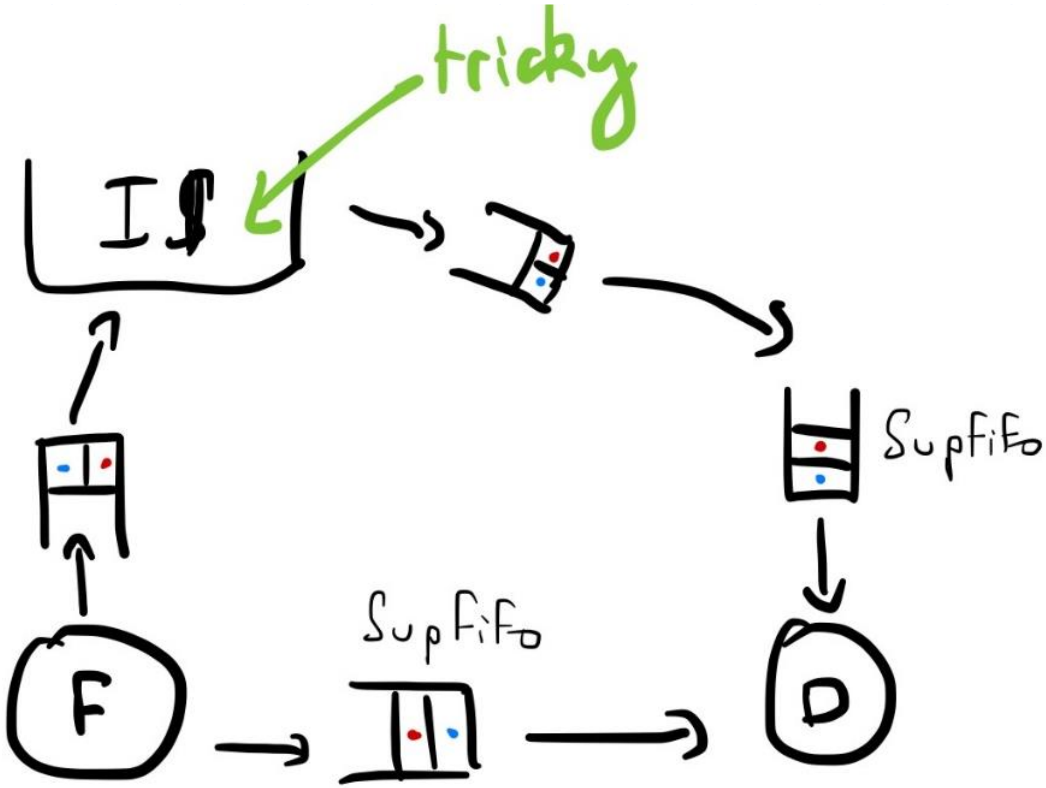
Extra difficulties with **S**MT

- Fetching two different lines from the I$ the same cycle?
  - Requires a 2-ported BRAM/SRAM
- What if we don’t have those for our I$? ☹
  - We can play a little trick!

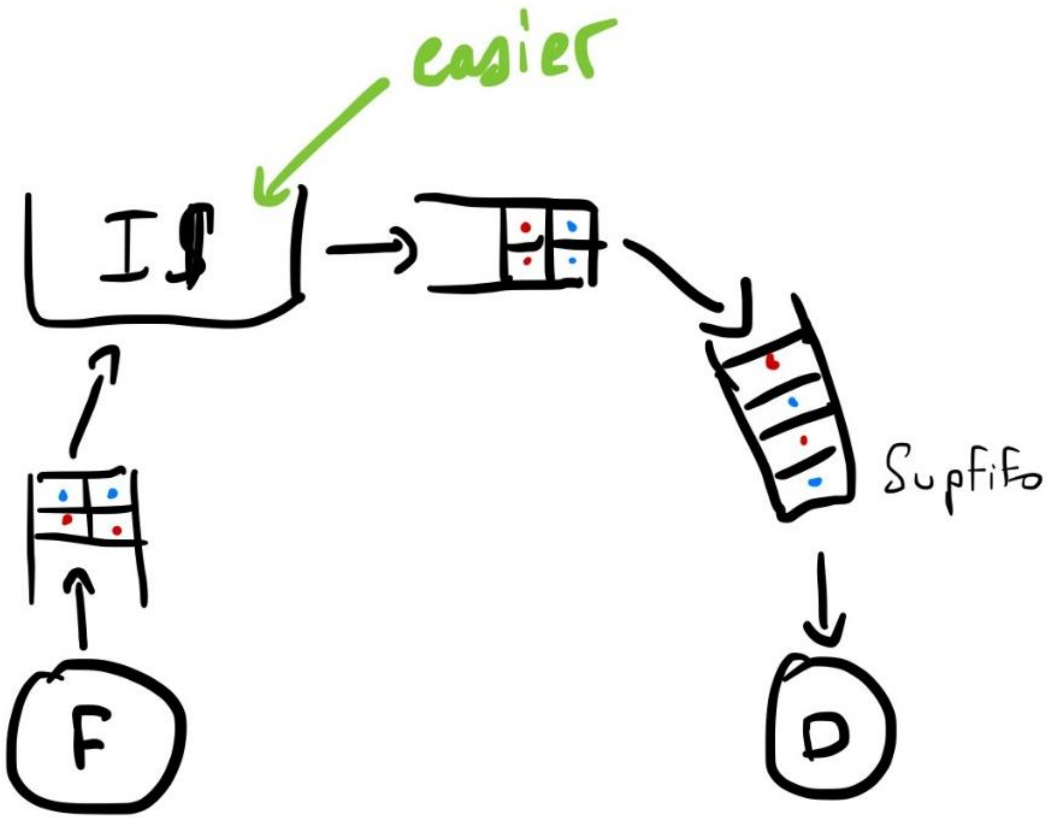
L1 superscalar



L1 for SMT - 2-port I$



L1 for SMT – 1-port I$



Scope, gains, losses?

1. We showed 2-way multithreading, does it works for k-way (k=3,4…)?
   - Modern intel: k=2
   - Alpha Ev8 (Araña): k=4
   - GPU ?
2. How much performance is there to win?

Let’s read Wikipedia together!

The elephant in the room:

What is a thread?

HW thread vs SW thread

- Scenario, two programs, one pc register:
  - swth0 runs for 1ms
  - Time … shelv…
    - OS … eturn to current pc, or to another pc, maybe pc of swth1
  - swth1 runs for 1ms
- Multiple SW thread can be run with one HW thread

Only a few extra registers are necessary to run multiple threads on a single hardware thread

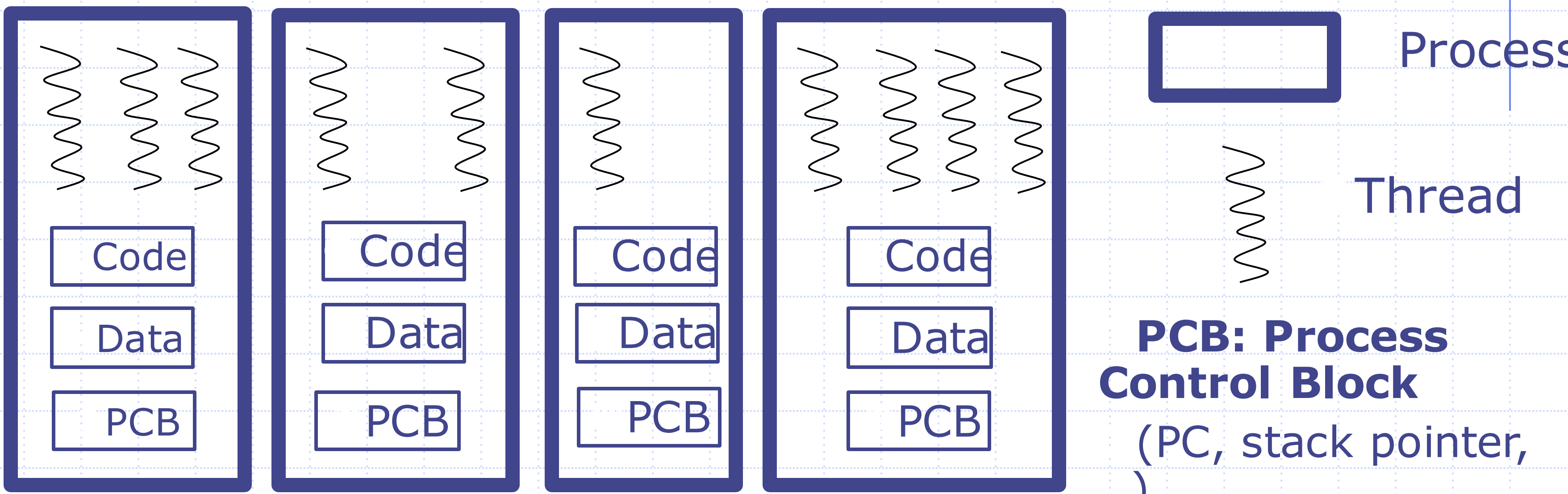
What we did not talk about

- With current setup:
  - The two programs share the same memory
  - They can mess with each other memory
  - BAD the 2 programs run by our HW threads are conceptually completely different
- Side-Quest: Could we provide an abstraction?

Virtual memory idea

- Addresses of the program are "virtual"
  - The hardware add one level of indirection to actually find the real address
  - The translation depends on the thread
- SW Thread VS Process:
  - Two threads of the same process, have the same translation

Processes and threads



PCB: Process Control Block
(PC, stack pointer, ..)