
EO-243: Computer Architecture

Instructors:
Arkaprava Basu
R. Govindarajan

Course Outline

- Processor Architecture, Caches, Instruction-Level Parallelism, Superscalar & VLIW architecture
- Multithreading, Multi-core processors, Cache Coherence and Memory Consistency; Multi-core Cache Organization
- Virtual Memory System
- Memory, DRAM Architecture
- Performance Evaluation
- Data-level Parallelism and Accelerators
- Power management
- Datacenter architecture
- Architecture Support for System Security

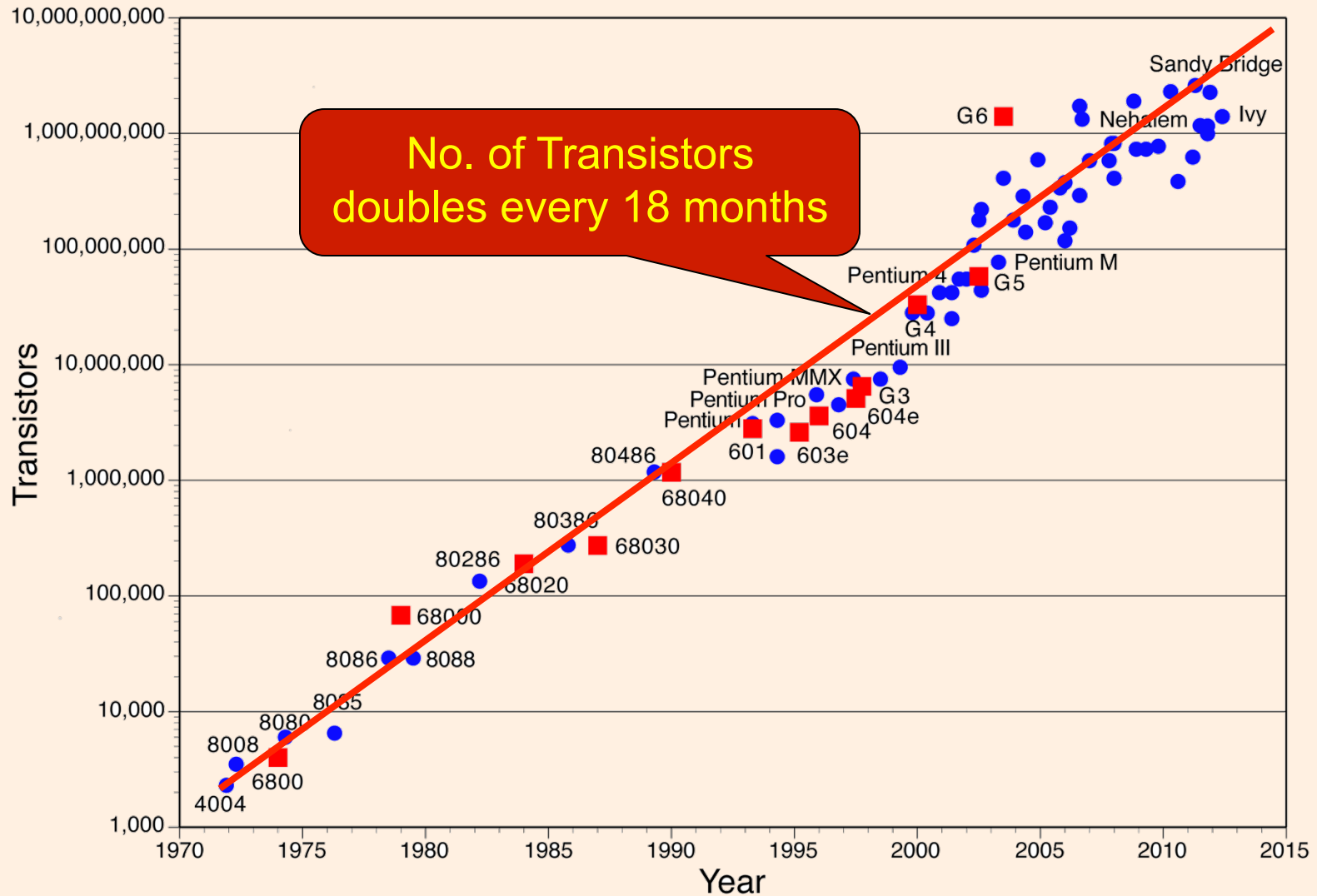
Other Details:

- **Lecture Hours:** M W (F) 11.00 -12.30 Hrs.
- **Text:** Hennessy and Patterson, "Computer Architecture: A Quantitative Approach":
 - 6th Ed. (Available in Indian Edition!)
 - A large part of it would be assumed as reading exercise!
- **Papers:** Announced during the course (Expect to have a fair amount of self-reading!)
- **Webpage:**
www.csa.iisc.ac.in/~arkapravab/courses/ca_f19.html
- **Course Interaction:**
 - www.piazza.com

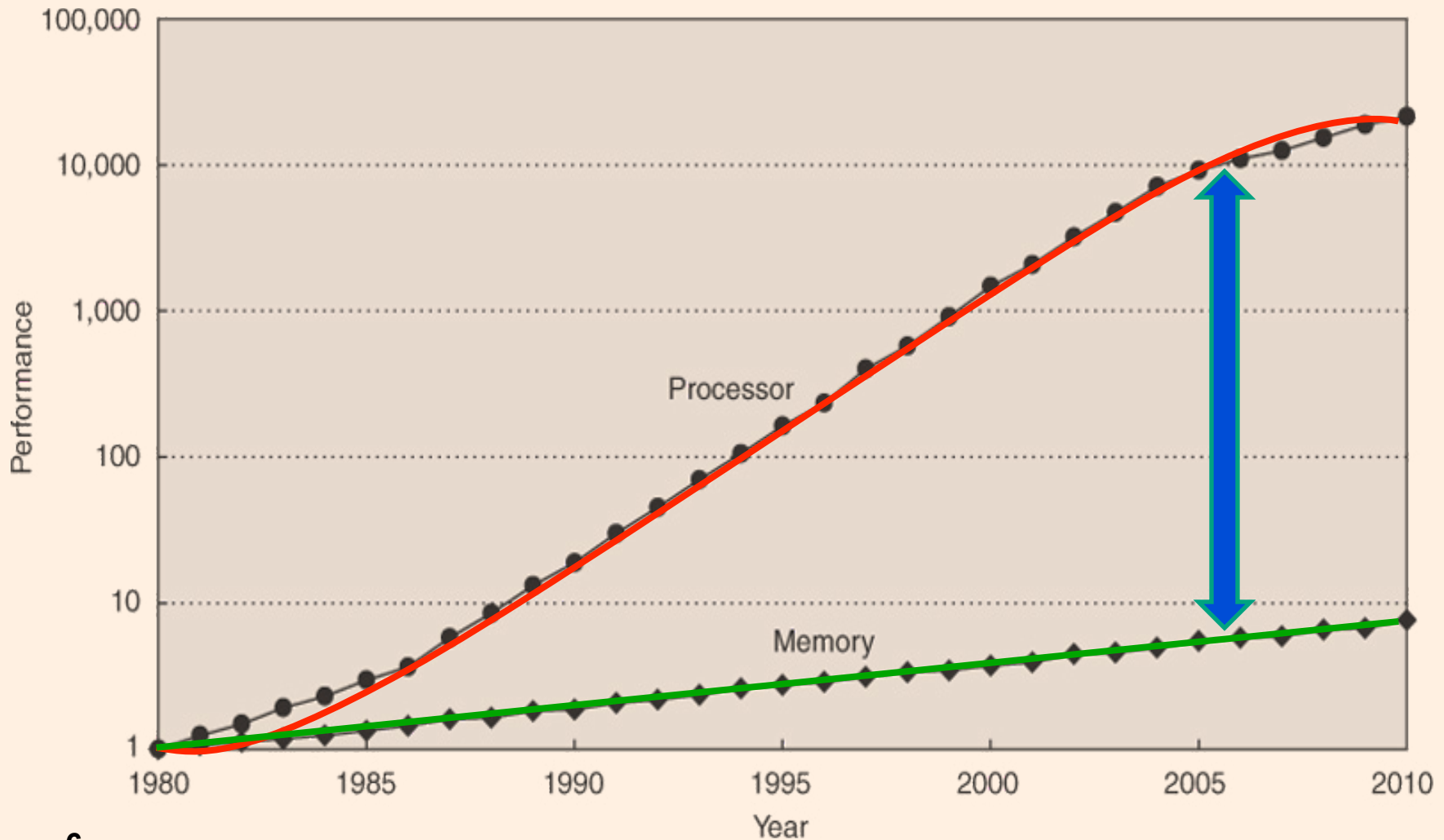
Grading

- 1 Mid-Term Test = 20 marks
 - Sept. 2nd/3rd week
- 1 Assignment = 10 marks
 - Sept. 1st week
- 1 Term Project = 30 marks [Group of 2]
 - Proposal Due by Sept. 15
 - First Review by Oct. 15
 - Report and Demo/Presentation : Nov. 30
- 1 Final Exam = 40 marks

Moore's Law



Moore's Law : Other Implications



Amdahl's Law

Speedup is limited by the part of the program which doesn't benefit by an enhancement

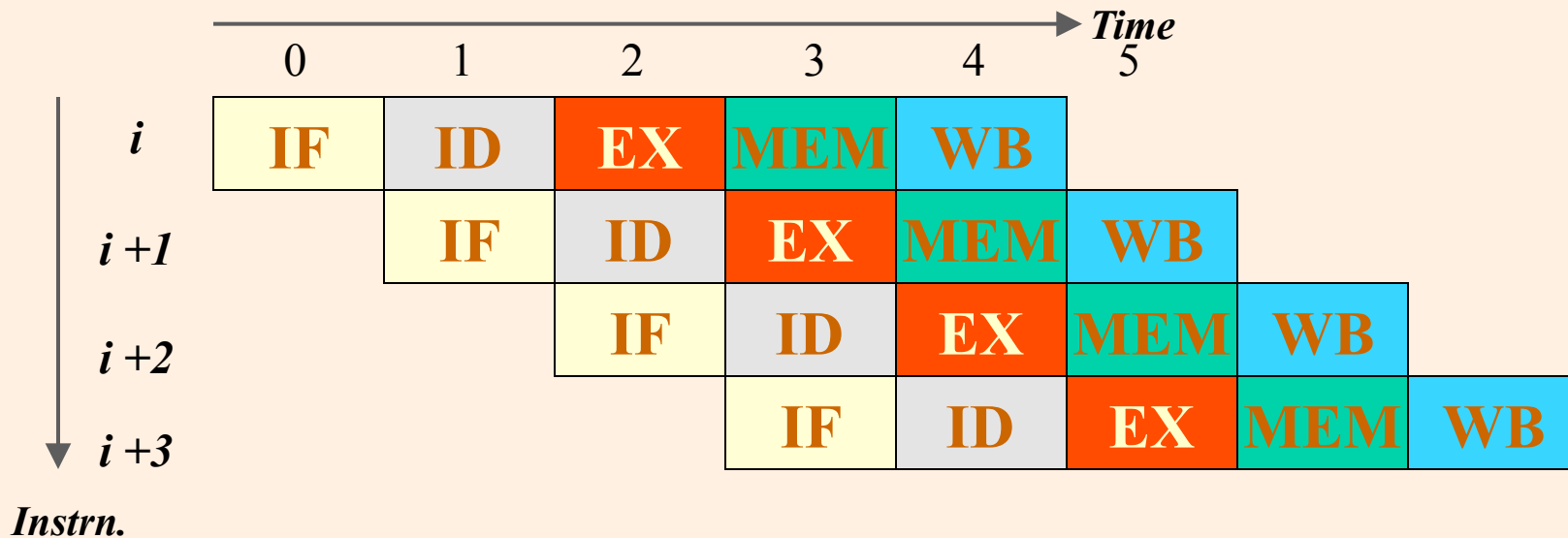
$$Sp = \frac{1}{s + (1-s)/k} = \frac{1}{s}, \text{ for large } k$$

Example : $s = 0.5$, $k = 100$, $Sp = 1.81$
Maximum Speedup is 2!

Moral : Make the common case faster!

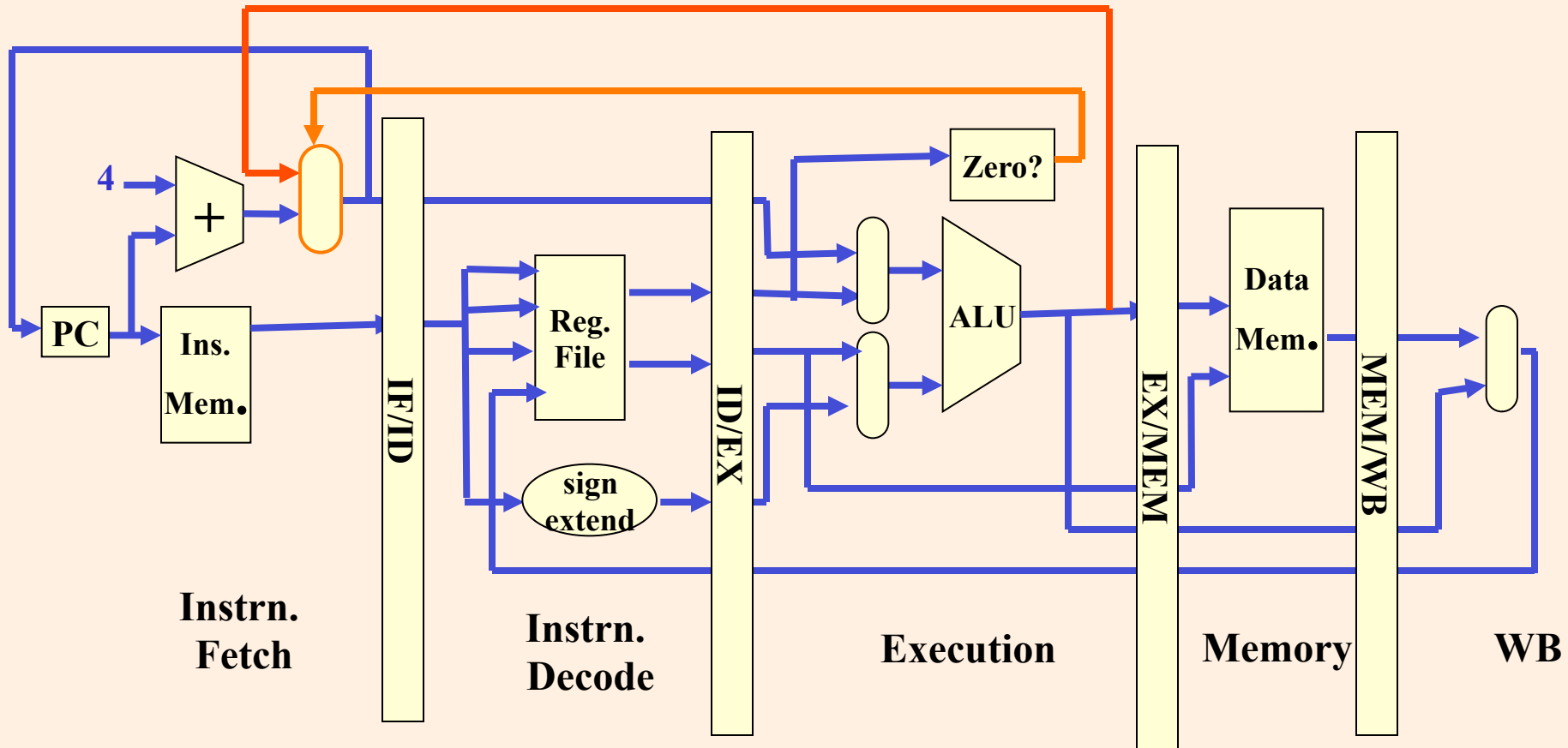
Pipelining : Review

Pipelining



- Exec. Time of instrn. still 5 cycles, but throughput, now is 1 instrn. per cycle.
- Initial pipeline fill time (4 cycles), after which 1 instrn. completes every cycle

Pipelined Processor Datapath



Pipeline Hazards

- Hazards prevent next instruction from executing during its designated clock cycle

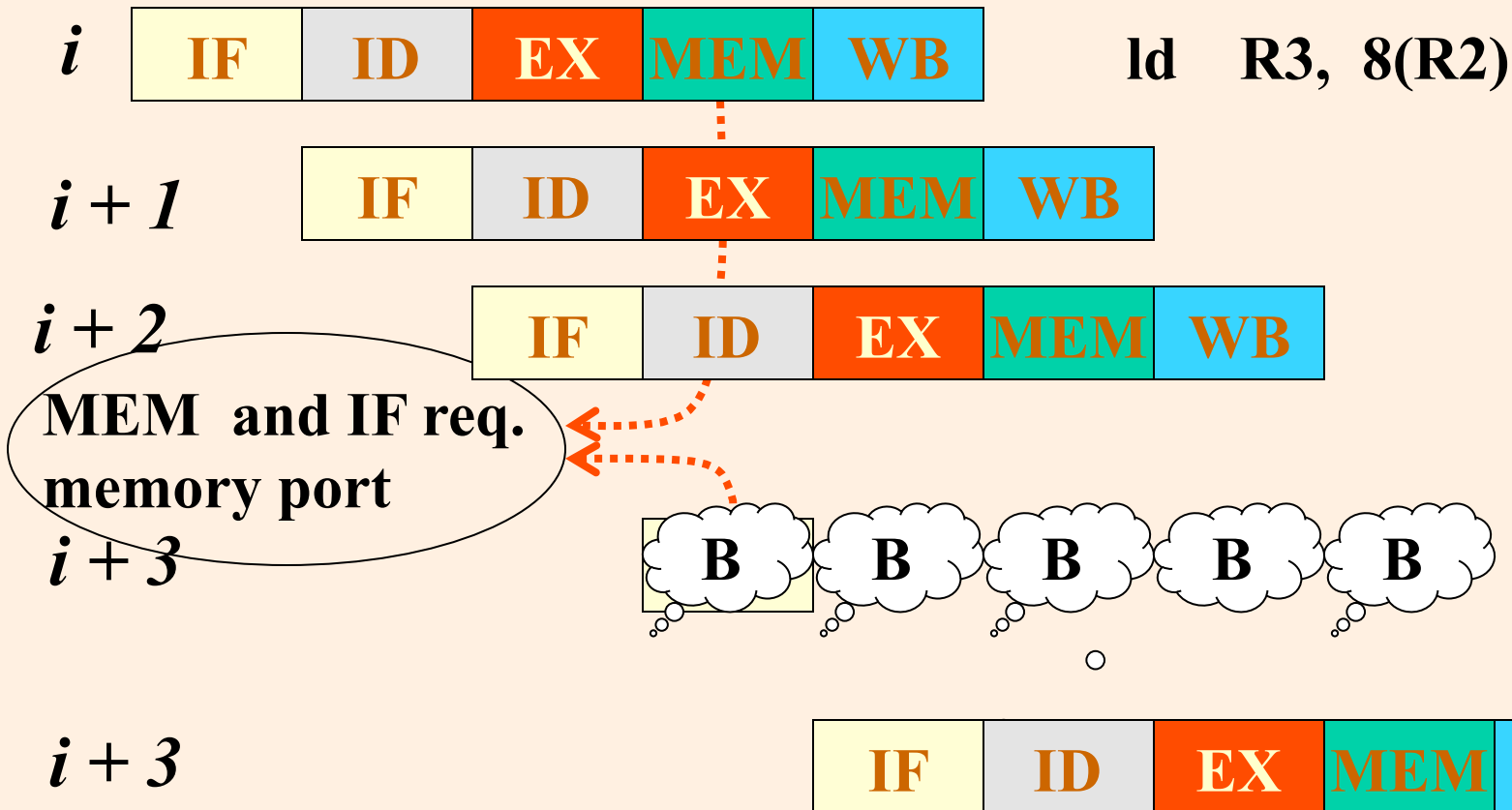
Structural hazards: Happen due to simultaneous request for the same resource by 2 or more instrns. (e.g., IF and MEM both require memory port!)

Data hazards: Instruction depends on result of prior instruction still in the pipeline.

Control hazards: Due to branch and jump instrns. Next PC (target PC) available only after 3 cycles (in EX stage) while IF has to take place in the next cycle!

Structural Hazard

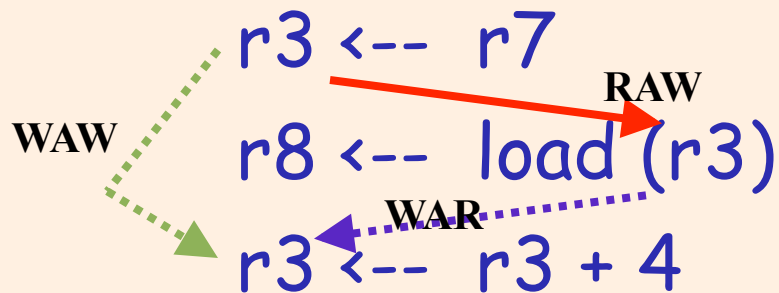
Assume a single memory port



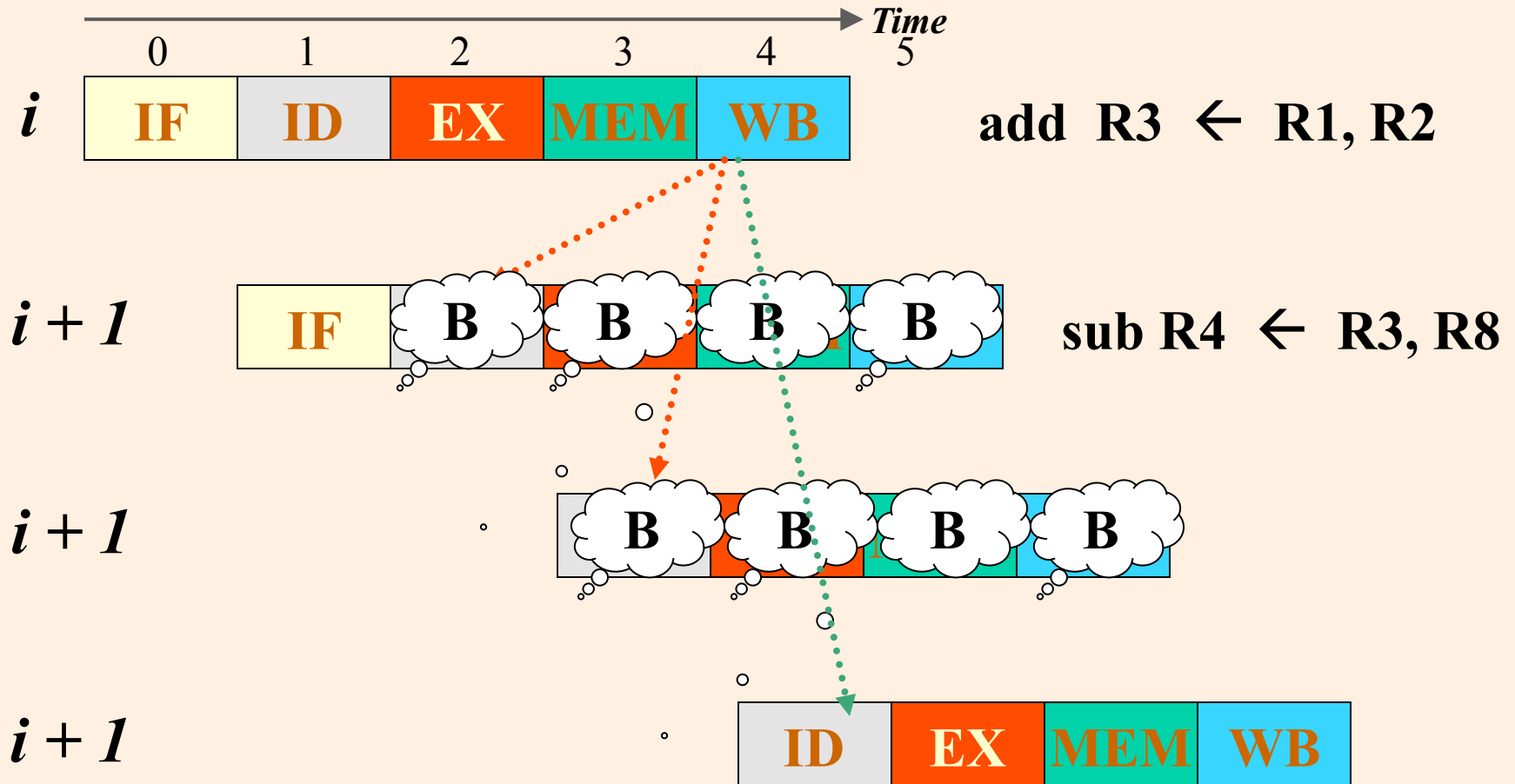
Data Dependence

- True data dependency (RAW)
- Anti dependency -- write-after-read (WAR)
- Output dependency -- write-after-write (WAW)

Example:



Data Hazard



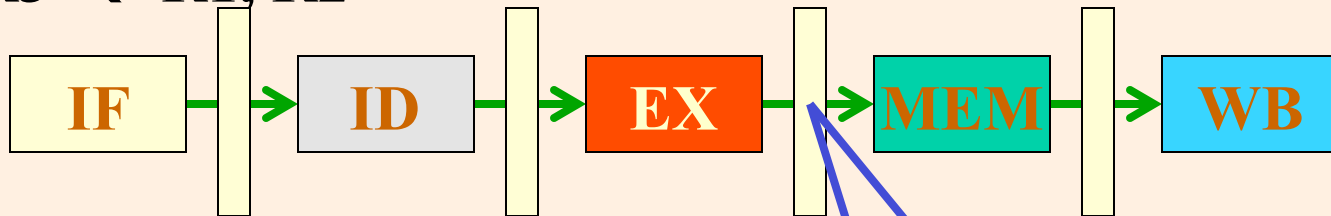
Data Hazard Solutions

- **Interlock:** Hardware detect data dependency and stalls dependent instrns.
- **Byeassing or Forwarding:** Computed data forwarded as soon as available (from EX or MEM stage)

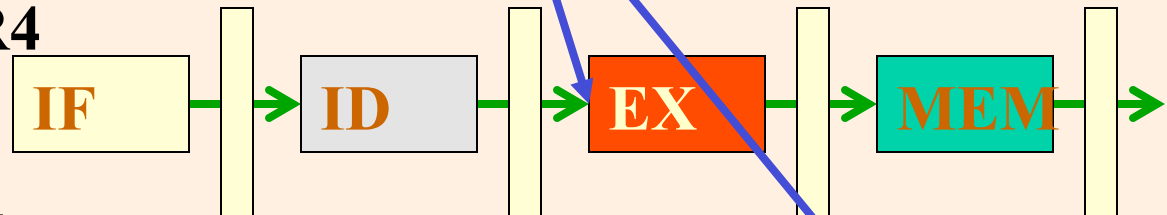
Data Hazard Solutions (contd.)

- Forwarding or Bypassing : forward the result as soon as available (EX or MEM stage) to EX.

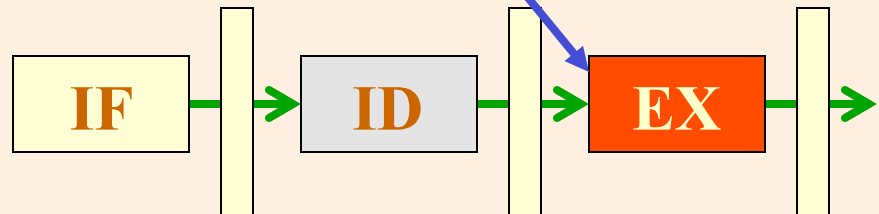
add R3 \leftarrow R1, R2



sub R5 \leftarrow R3, R4



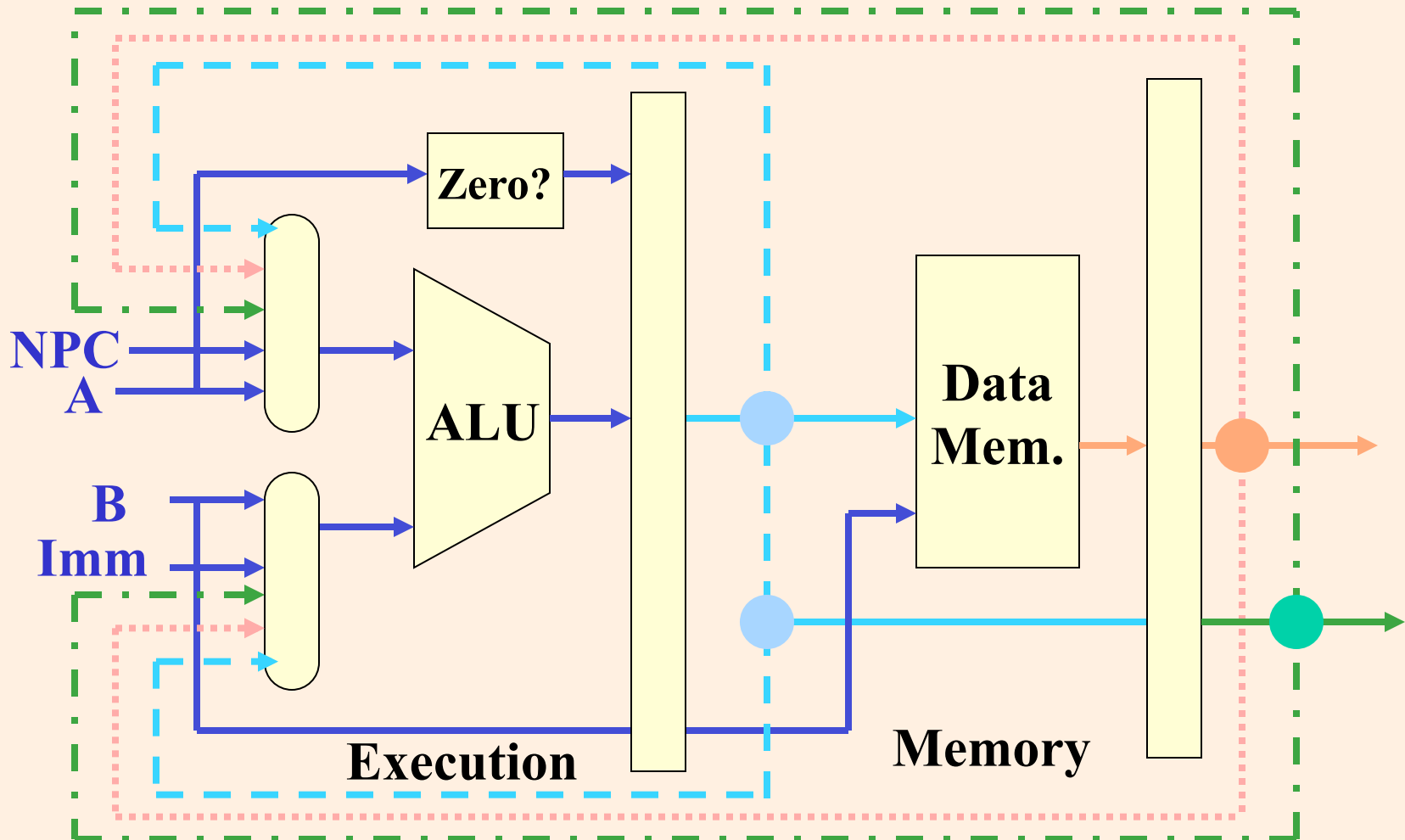
or R7 \leftarrow R3, R6



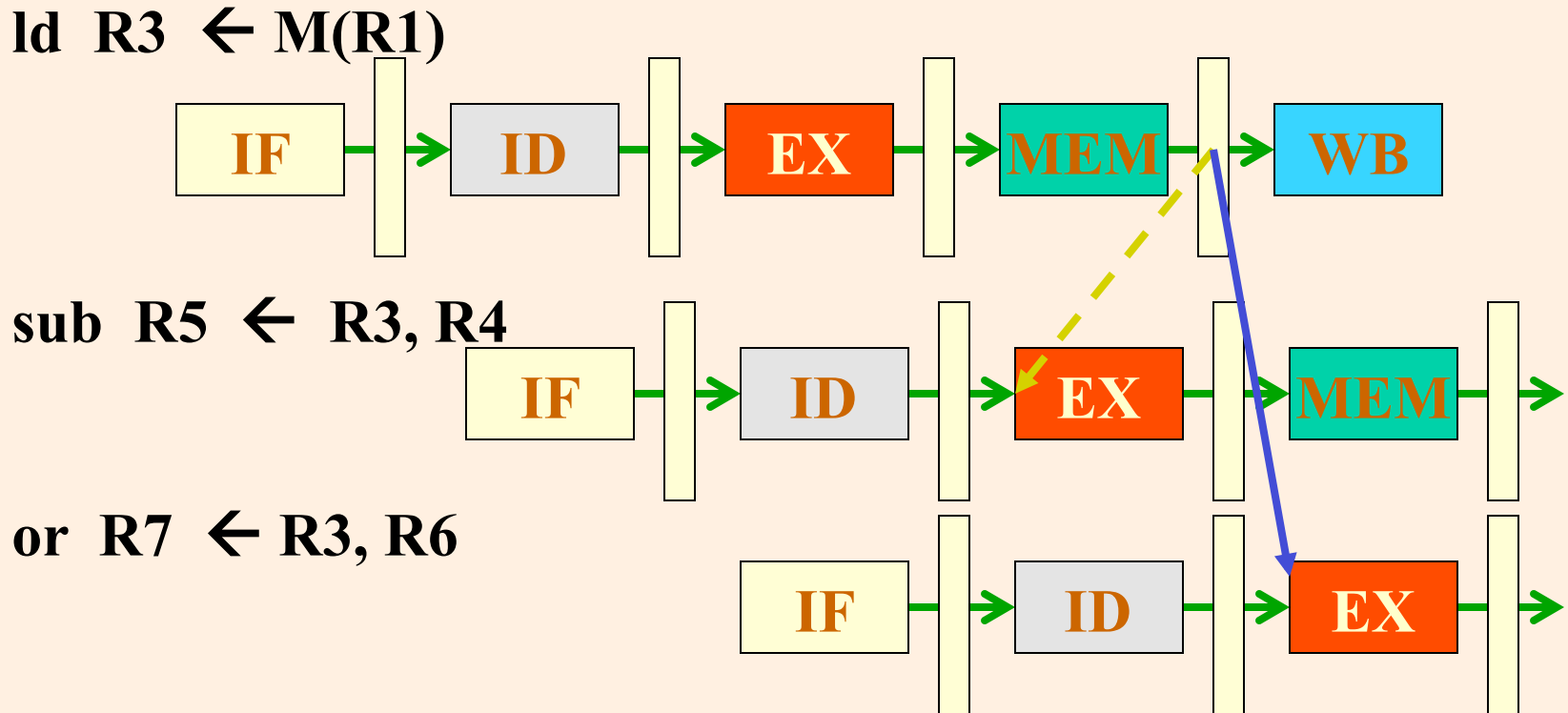
Data Hazard Solutions

- **Interlock:** Hardware detect data dependency and stalls dependent instrns.
- **Byeassing or Forwarding:** Computed data forwarded as soon as available (from EX or MEM stage)
 - For what Ops from MEM stage?
 - Can byeassing avoid all stalls?
- **Instruction Scheduling:** Reorder instrns. such that dependent instrns. are 2-3 cycles apart.
 - Static Instruction Scheduling
 - Dynamic Instruction Scheduling

Processor Datapath for Forwarding

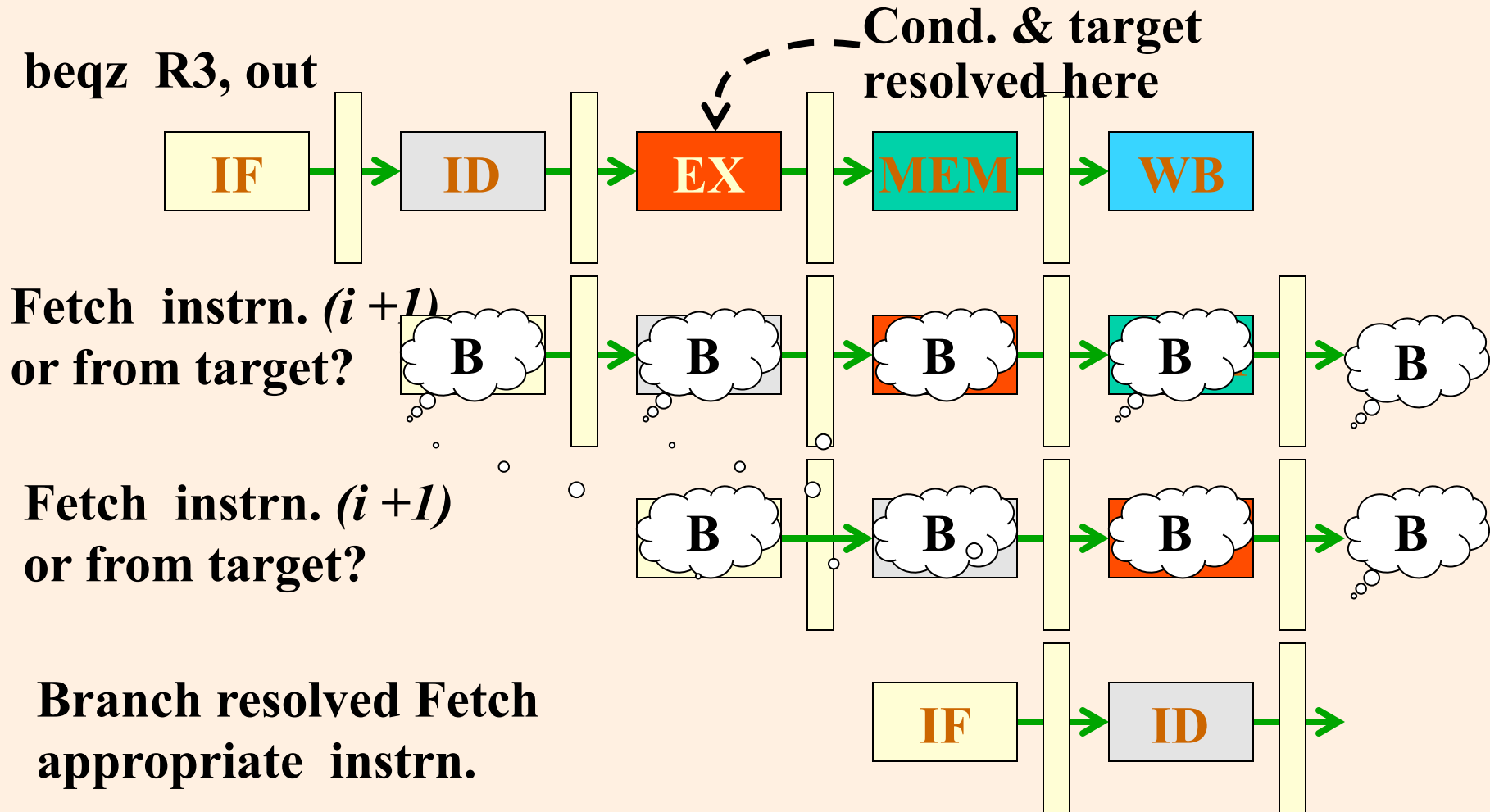


Forwarding may not always Work!

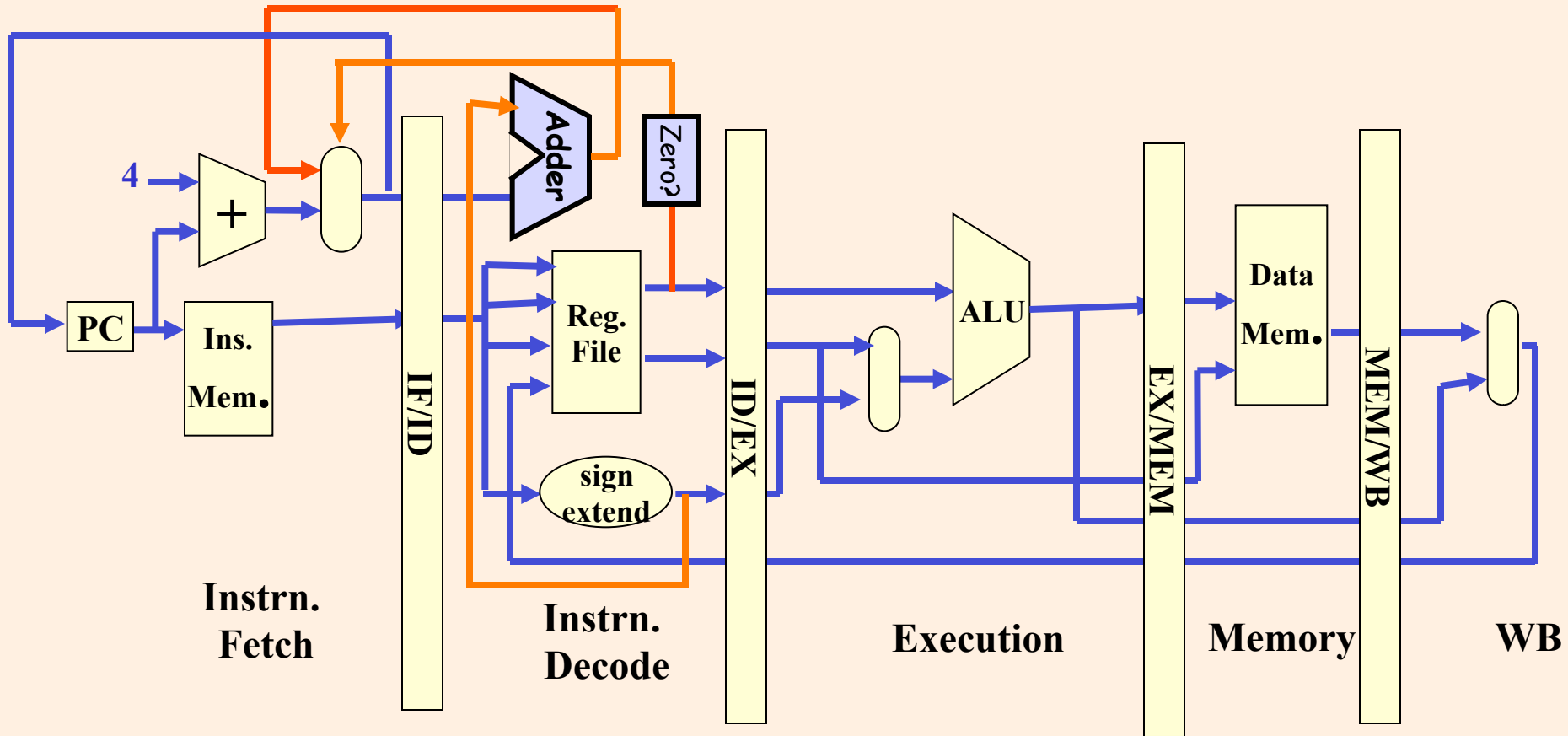


- Forwarding not very useful in deep pipelines!

Control Hazard



Pipelined MIPS Datapath



Control Hazard Solutions

- Static Branch prediction policies:

- Static Not-Taken policy:

- Fetch instrn. from PC + 4 even for a branch instrn.

- Squash the fetched instrn, and re-fetch from branch target address

- Delayed Branching:

- microarchitecture designed such that branch (control transfer) takes place *after* a few following instructions.

```
beq R1, out
```

```
add R3 <-- R2, R4
```

(delay slot instrn.)

Delayed Branching

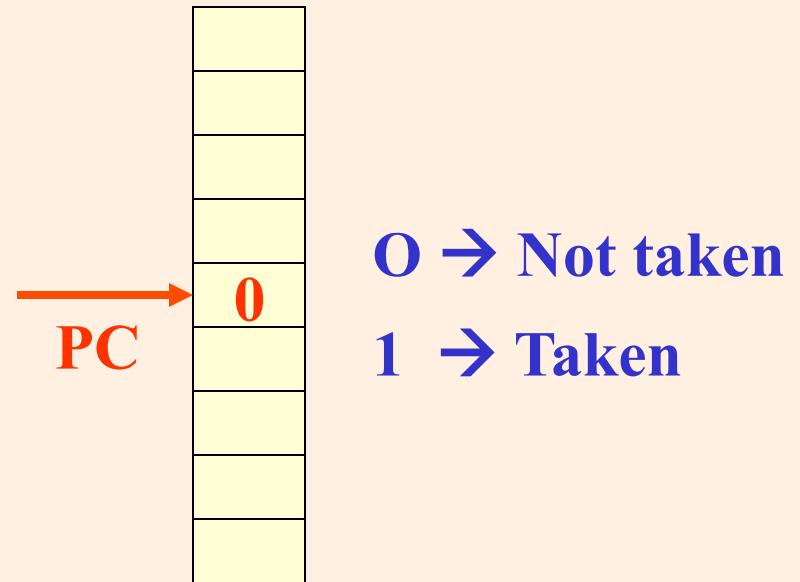
- Instrns. that do not affect the branching condition can be used in the delay slot.
- Where to get instrns. for filling delay slots?
 - Instrns. Before the branch
 - Instrns. from the target address
 - Instrns. from fall through
- Cancelling delayed branches, which squash the instrn. in the delayed when the branch is taken (or not taken) -- facilitate more slots to be filled.

Dynamic Branch Prediction

- For the same static branch instrn. prediction for different instances may be different.
- Using the history of branches to predict the branch outcome (taken or not-taken)
 - 1-bit predictors
 - 2-bit predictors
 - 2-level correlated predictors
 - path-based predictors, ...

Simple 1-bit Branch Prediction

- Store previous history of a branch -- taken or not-taken (1-bit) in Branch Prediction Buffer
- Least Significant bits of PC used to index BPB.
- Update Predict bit after branch is resolved.
- Predict branch outcome only, not target address.
- Influence of other (static) branches which have same LS bits (alias)

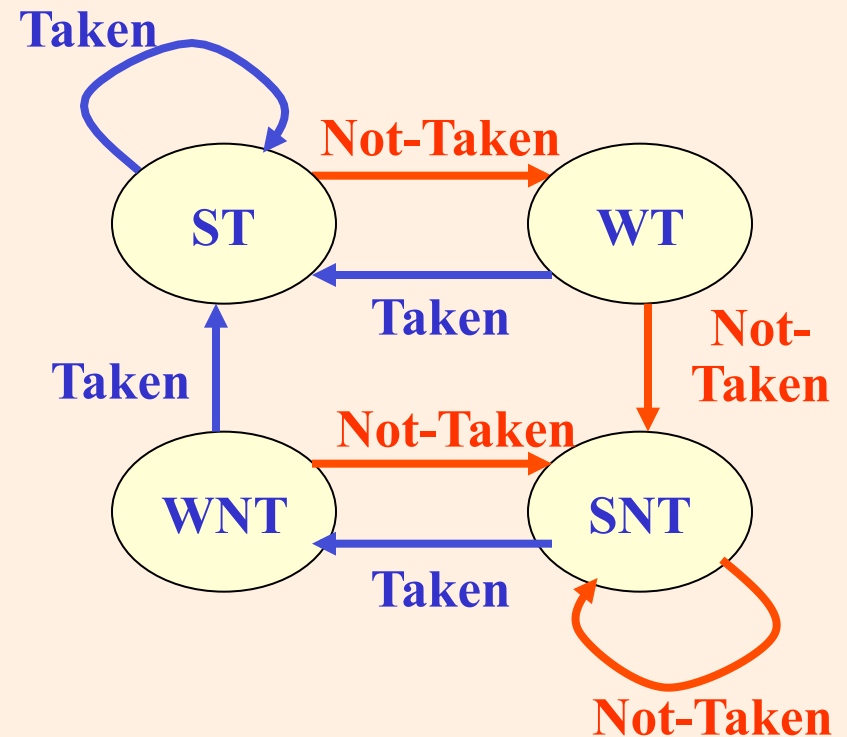


BPB (2048 entries)

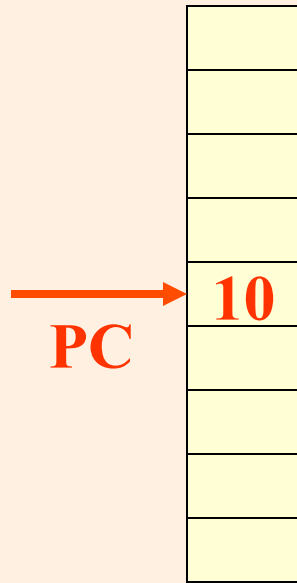
- 1-bit prediction will mispredict twice on every loop.

2-bit Branch Prediction

- State of each static branch represented by 2-bits.
- Implement the State machine
- Implementation using saturating counters ?
- Gives high prediction accuracy 85% to 95%.
- Different state transition diagram for diff. 2-bit schemes.
- What is the benefit of 2-bit over 1-bit?



2-bit Prediction



00, 01 → Not taken
10, 11 → Taken

BPB (2048 entries)

- Still no prediction of target address!
- With a 2-bit prediction the no. of mispredictions of branch b1 is lower!

L1: R2 ← 100

...

L2: R1, ← 100

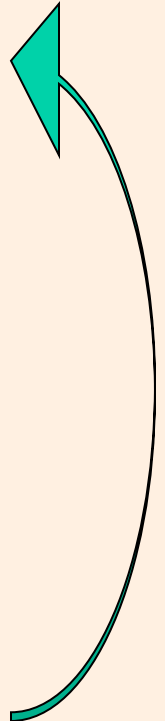
...

R1 ← R1 - 1

b1: bne R1, L2

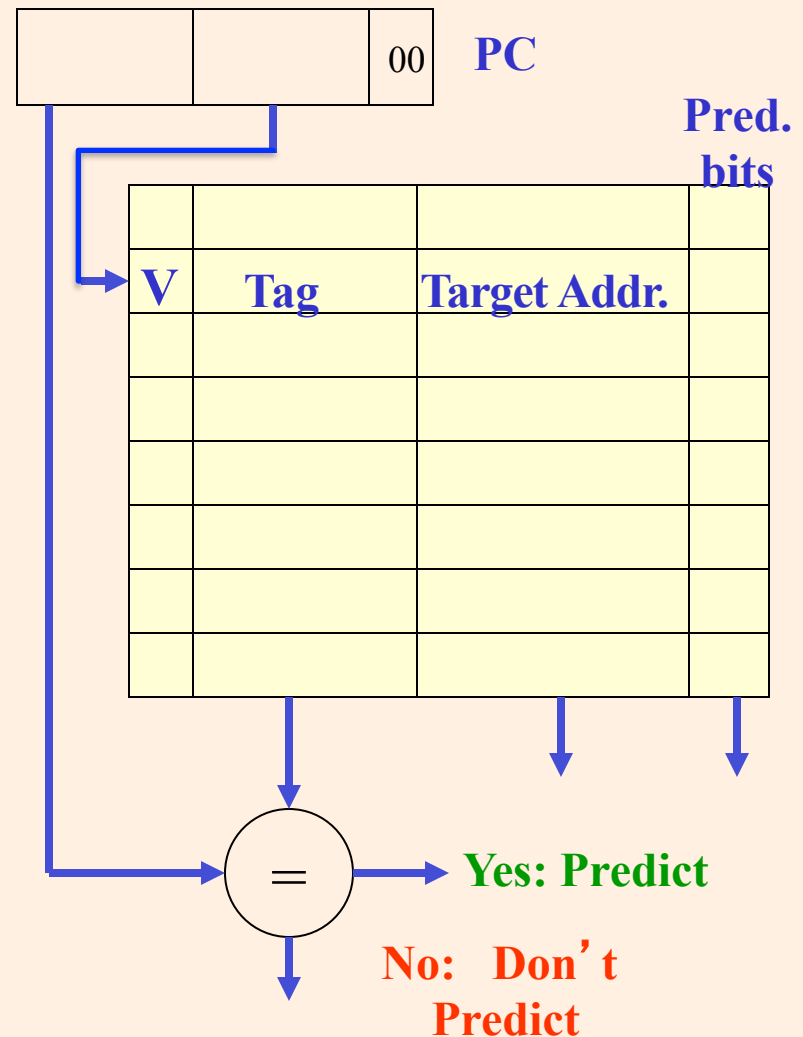
R2 ← R2 - 1

b2: bne R2, L1



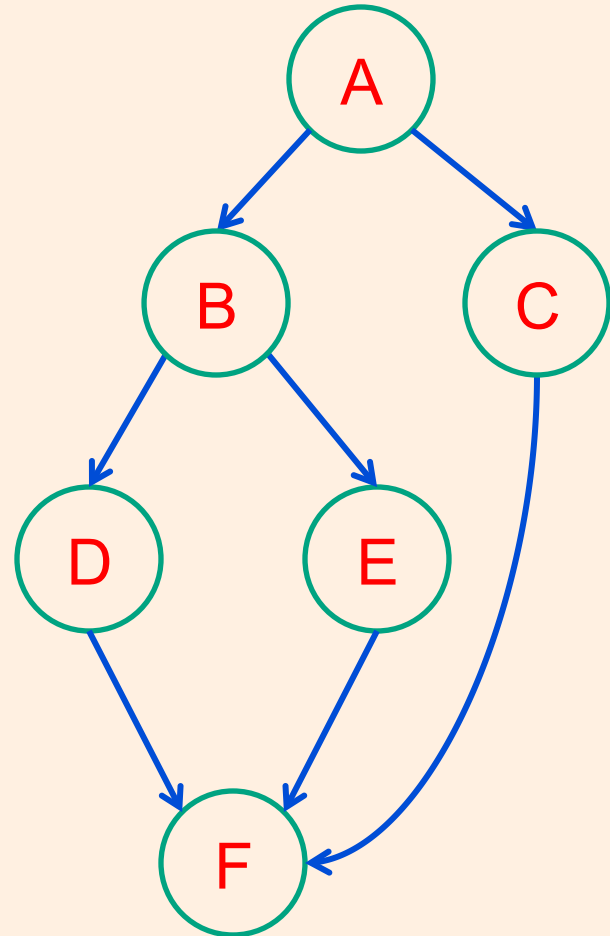
Branch Target Buffer

- Store branch target address (computed when the static branch instrn. is encountered first) along with prediction bits -- Branch Target Buffer.
- Associate tag bits to avoid other (branch) instrns. influence the prediction.
- When to make the prediction?
 - Prediction can be made in IF stage itself! **no stall** on correct prediction.
- Size of BTB = 2048x52 !



Branch Prediction (contd.)

- What other alternative prediction scheme can be used?
 - Correlation between branches?
 - Predicting the path rather than the branch?



2-Level Branch Prediction

- 2-bit scheme uses the history of the same static branch for prediction.
- In practice, history of other branches also influence branching.

```
if (aa == 2) aa = 0;  
if (bb != 2) bb = 0;  
if (aa == 0) || (bb == 0) { ...
```

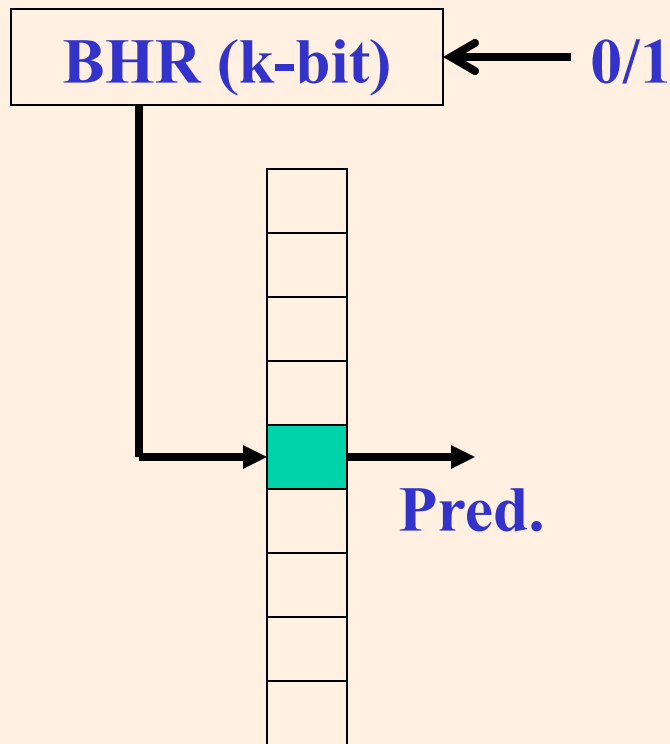
history	br1	1	1	0	0
	br2	1	0	1	0
Pred.	br3	1	1	1	?

2-Level Prediction (contd.)

- Branch history register/table to maintain the outcome of previous (dynamic) k branches.
- Pattern History table(s) to record the branch behavior (2-bit FSM).
- Branch history can be global or local
- Pattern history can be global or local.
- Other variations (g-share, p-share)
- Results upto 98% prediction accuracy.
- **Reading:**
 - TseYu Yeh and Yale N Patt, "Alternative Implementations of TwoLevel Adaptive Branch Prediction", ISCA 92

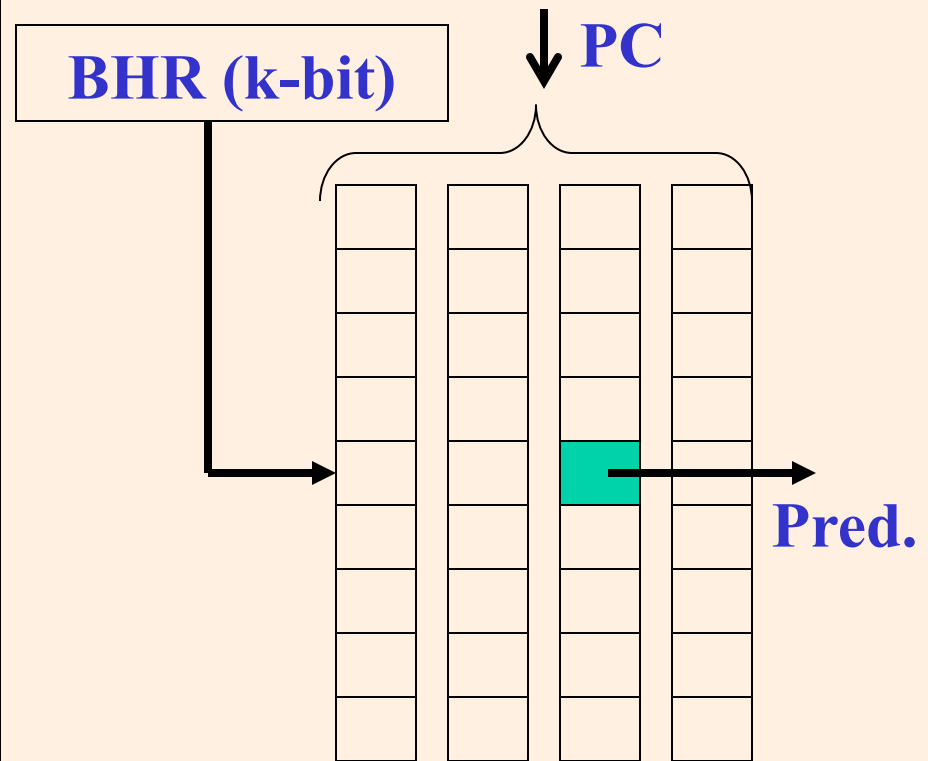
GAg and GAp Schemes

Global branch history register & global pattern history table



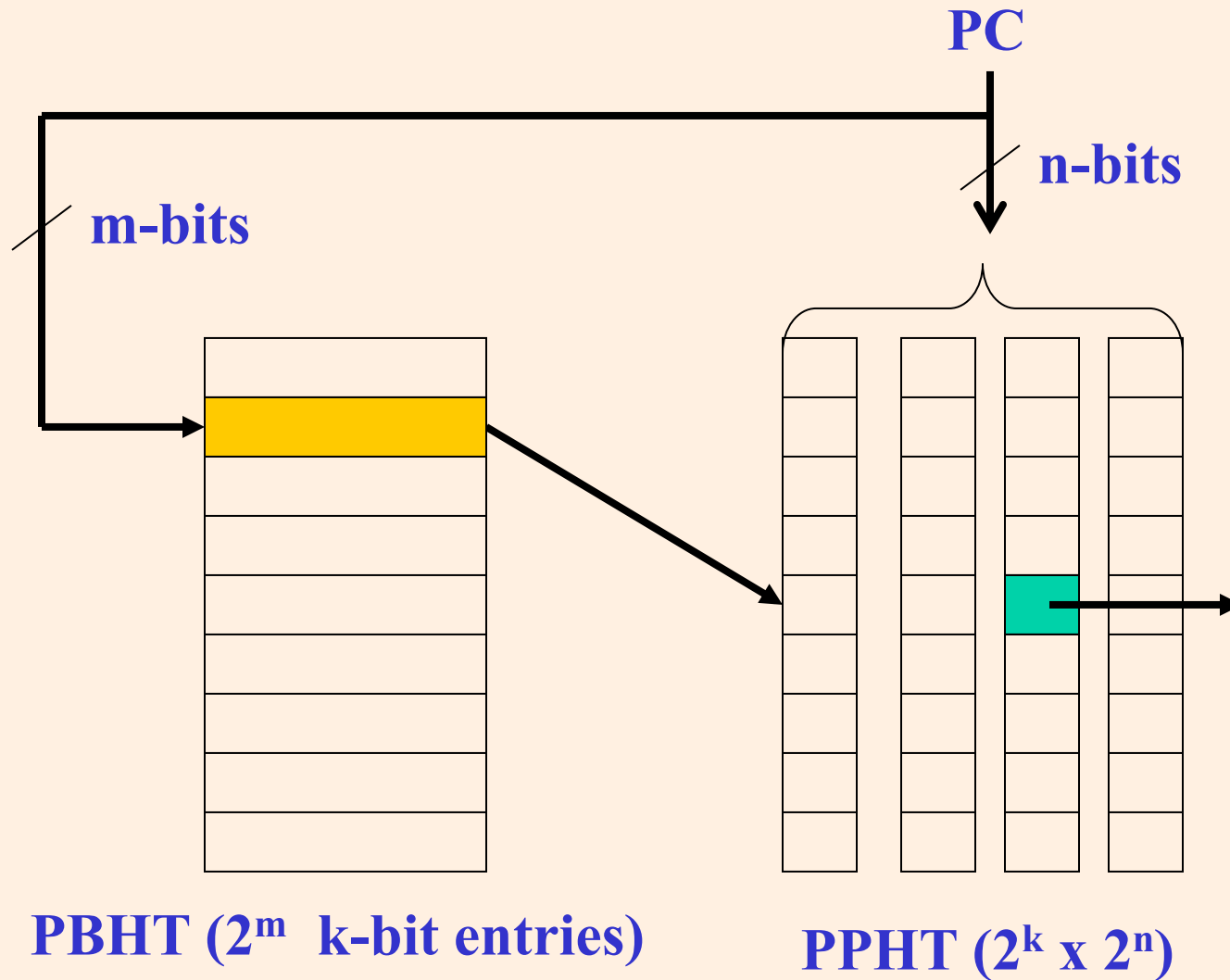
PHT (2^k entries)

Global branch history reg. and per addr. pattern history table



PPHT ($2^k \times 2^m$)

PAp Scheme



Comparison of 2-Level Schemes

- In *G*Ag, 1st-level history is based on last *k* dynamic branches.
 - No per-addr. Branch or pattern history.
 - History of all branches influence prediction.
 - In *G*Ap, correlation of other branches, and past behavior of this branch influence prediction.
- In *P*Ag, per-addr. branch history is maintained.
 - History of a branch influence its prediction; but same pattern behavior for all branches
 - *P*Ap other branches influence is least.
 - Per-Set schemes use addr., opcode type, etc. to determine sets.

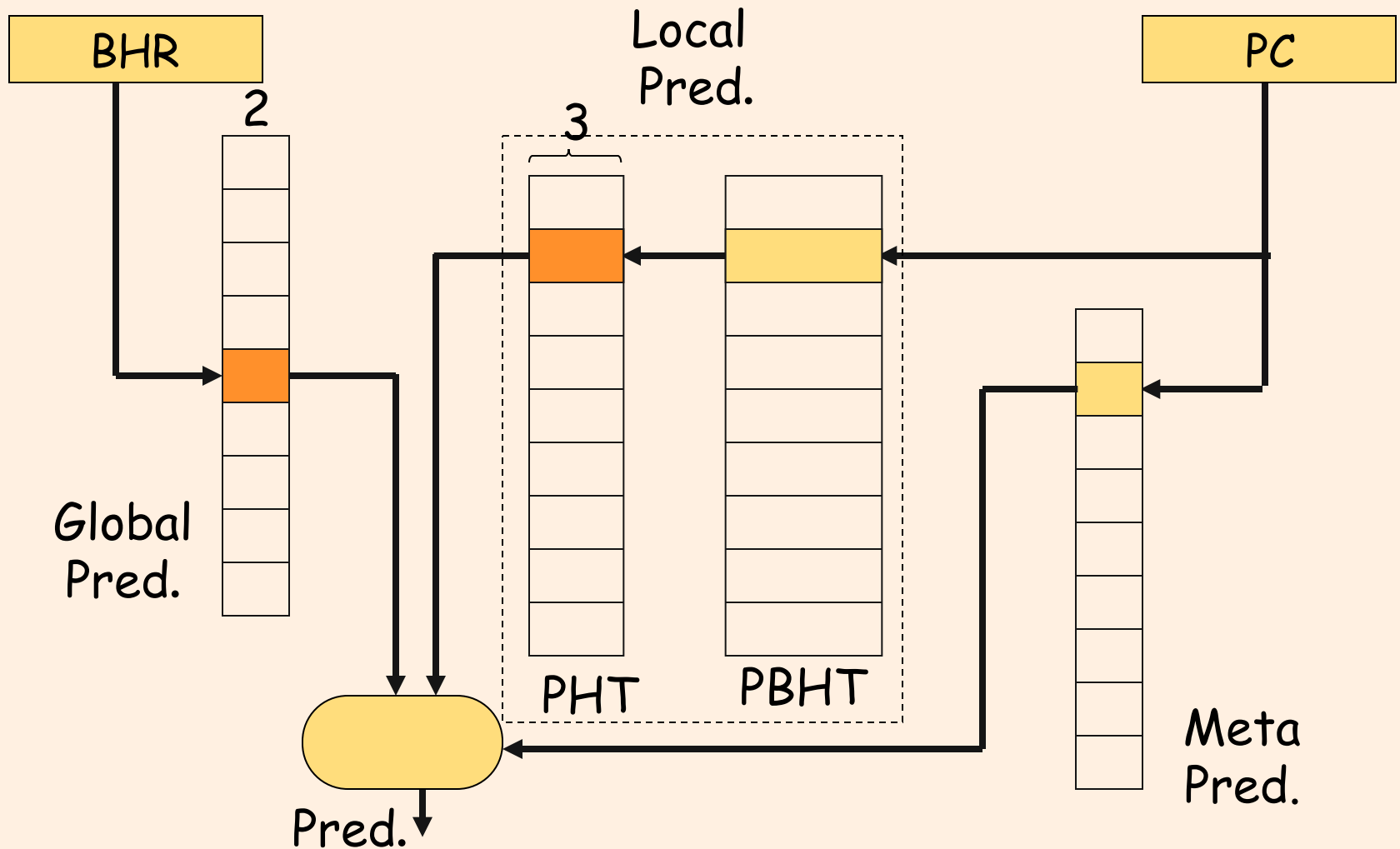
G-Share and Other Predictors

- G-share is similar to GAg, except that BHR is XORed with PC to index in PHT.
 - Eliminates interference of same branch history across different branches.
- 2-Level predictors have longer warm-up time to build history. Frequent context-switching affects building the history.
- Hybrid predictors take advantage of multiple predictors.
- Path-based schemes using history of target addresses of branches and not their outcomes.

Hybrid Predictors

- **Global predictor:** Has 4K entries and is indexed by the history of the last 12 branches; each entry in the global predictor is a standard 2-bit predictor
- **Local predictor:** Consists of a 2-level predictor:
 - **Top level** a local history table consisting of 1024 10-bit entries; each 10-bit entry corresponds to the most recent 10 branch outcomes for the entry.
 - **Next level** Selected entry from the local history table is used to index a table of 1K entries consisting a 3-bit saturating counters, which provide the local prediction
- 4K 2-bit counters to choose between the global predictor and local predictor
- Total size: $4K \cdot 2 + 4K \cdot 2 + 1K \cdot 10 + 1K \cdot 3 = 29K$ bits!
(~180,000 transistors)

Tournament Predictor in 21264



Indirect Branches

- Branches or jumps whose target addr. varies at runtime.
 - Function return, jumps for implementing case statements, etc.
 - Target address prediction for function returns using a **return address stack** of size 8 to 16.