

CS250P: Computer Systems Architecture

Processor Design Constraints



Sang-Woo Jun

Four large cores?
VS
Eight smaller cores?

How many registers
in register file?

17 pipeline stages
VS
40+ pipeline stages?

Given \$X,
how would you architect the best processor?

Simultaneous Multithreading?
Y/N?

N-way superscalar?

How many execution units
per pipeline?

Why?

Constraints involved in processor design



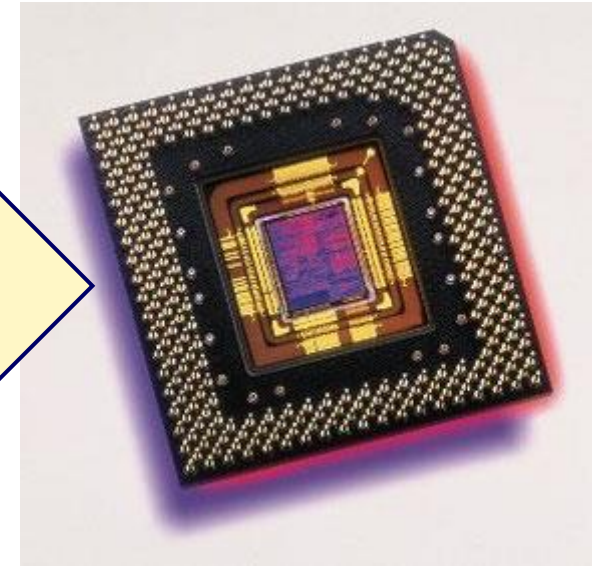
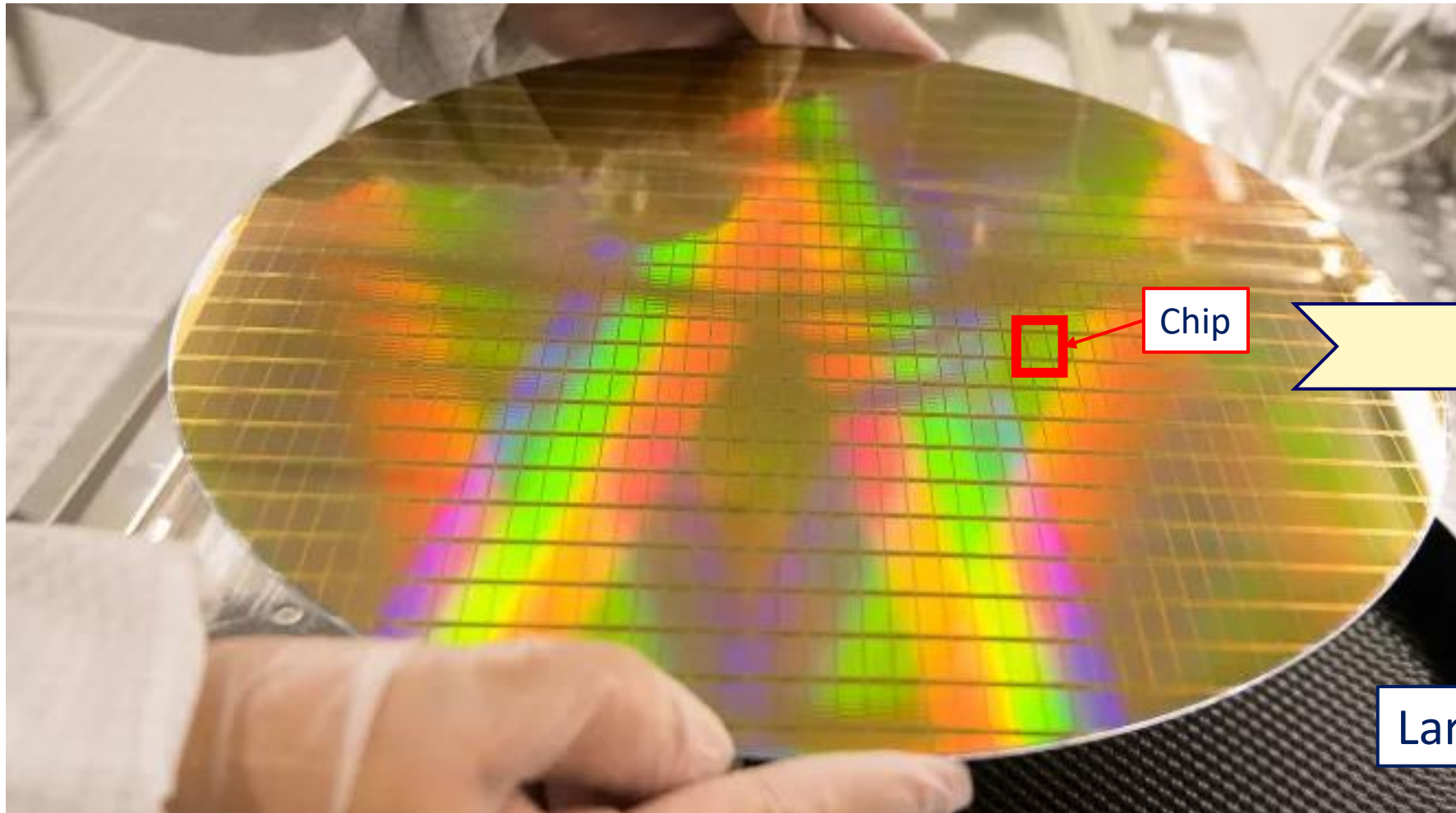
Chip Area

Attainable Clock Speed

Instruction-Level Parallelism

Amdahl's Law

Constraint #1: Chip Area per \$



Larger chip → Higher cost!

Constraint #1: Chip Area per \$

Moore's Law no longer leads to correspondingly cheaper chips per cost...

	16/12nm	10nm	7nm	5nm
Mass production year and quarter	2015 Q3	2017 Q2	2018 Q3	2020 Q
Capital investment per wafer processed per year	\$11,220	\$13,169	\$14,267	\$16,746
Capital consumed per wafer processed in 2020	\$993	\$1,494	\$2,330	\$4,235
Other costs and markup per wafer	\$2,990	\$4,498	\$7,016	\$12,753
Foundry sale price per wafer	\$3,984	\$5,992	\$9,346	\$16,988
Foundry sale price per chip	\$331	\$274	\$233	\$238

Increasing! 😞

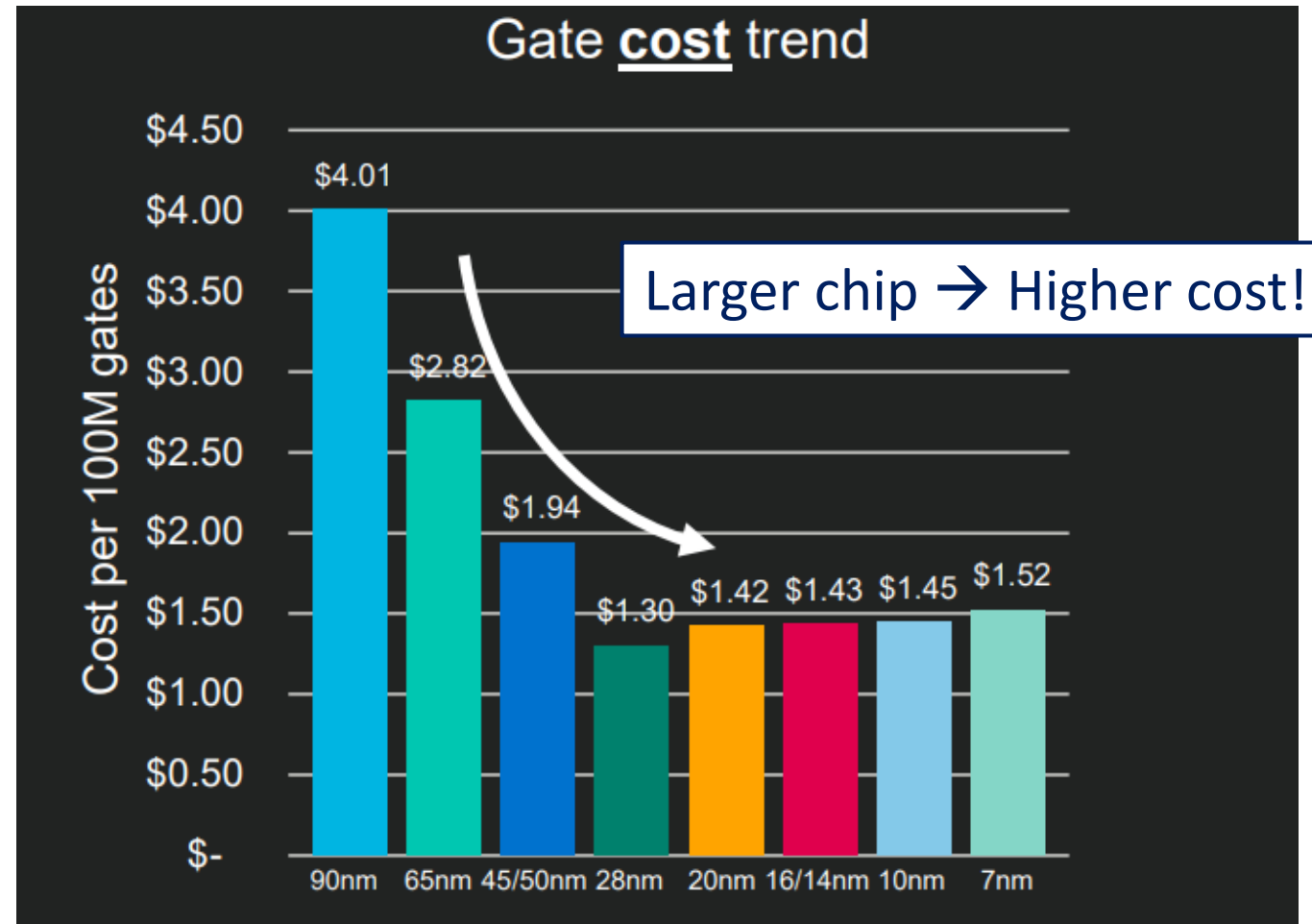
Steady...

Reproduced from CSET report "[AI Chips: What Are They and Why They Matter](#)"

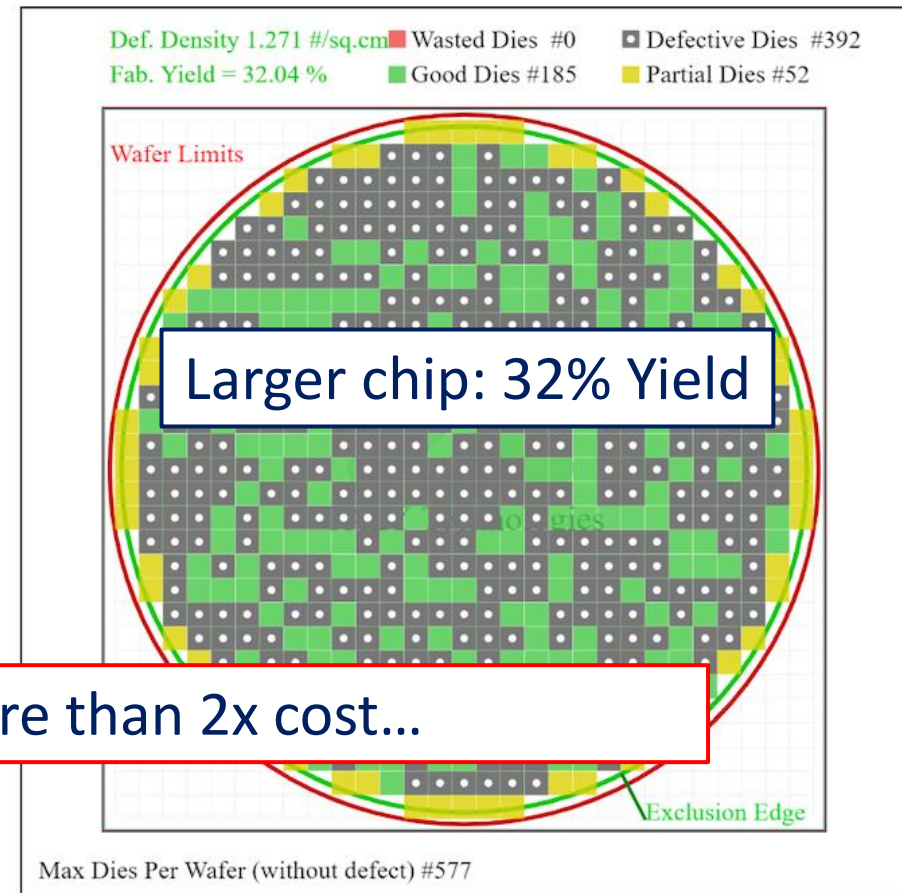
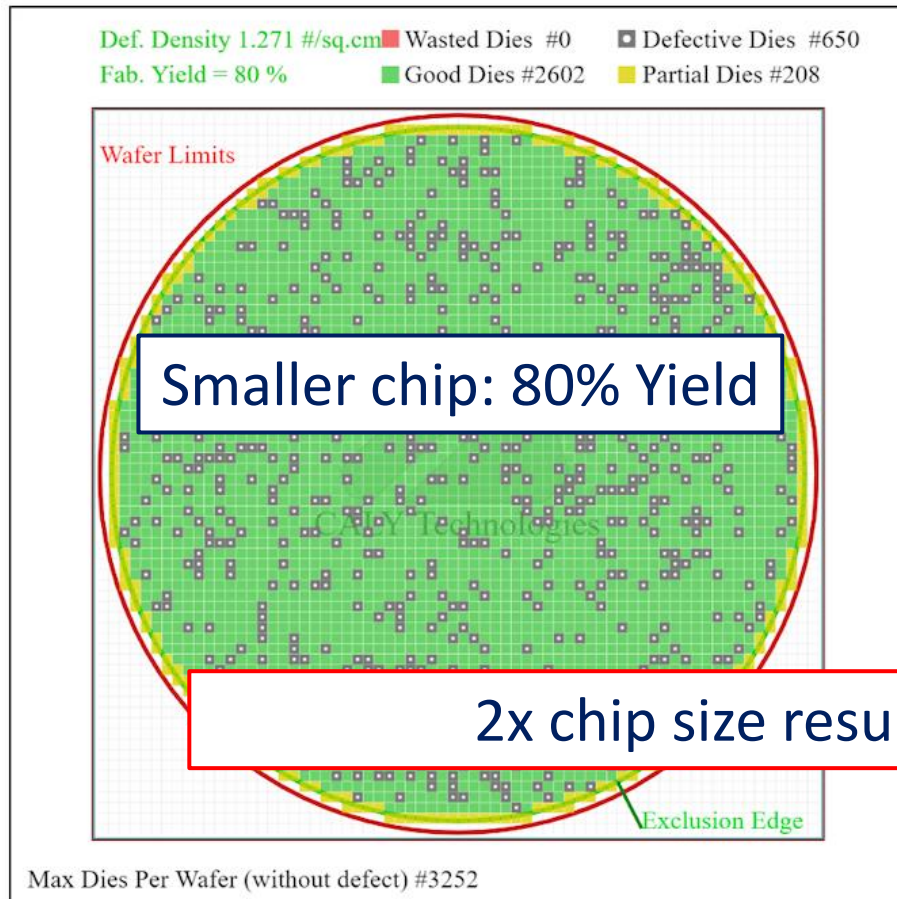
Constraint #1: Chip Area per \$

Gate	Delay (ps)	Area (μ^2)
Inverter	20	10
Buffer	40	20
AND2	50	25
NAND2	30	15
OR2	55	26
NOR2	35	16
AND4	90	40
NAND4	70	30
OR4	100	42
NOR4	80	32

Example standard cell library



Yield Plummets with Larger Chips ☹️



2x chip size results in more than 2x cost...

What is the cost/performance sweet spot?

After Considering Constraint #1

- ❑ The size of our processor may be limited...
 - How much chip space do we want to pay for?
- ❑ How fast can we make it go, within size limitations?

Constraints involved in processor design

Chip Area



Attainable Clock Speed

Instruction-Level Parallelism

Amdahl's Law

Constraint #2: Attainable Clock Speed

- ❑ “More complex ISA slows down the clock”

why?

This requires a bit of circuits recap...

Combinational and sequential circuits

❑ Two types of digital circuits

❑ Combinational circuit

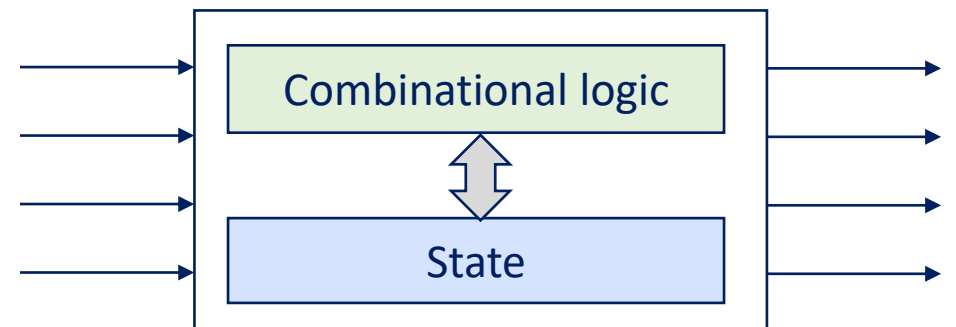
○ Output is a function of current input values

- $\text{output} = f(\text{input})$
- Output depends exclusively on input

❑ Sequential circuit

○ Have memory (“state”)

- Output depends on the “sequence” of past inputs

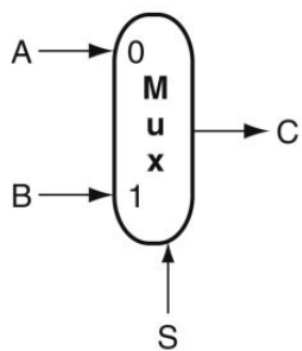


What constitutes combinational circuits

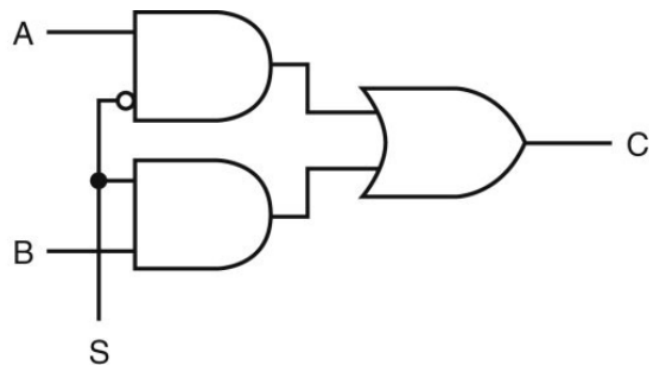
1. Input
2. Output
3. Functional specifications
 - The value of the output depending on the input
 - Defined in many ways!
 - Boolean logic, truth tables, hardware description languages, *We've done this in CS151*
4. Timing specifications *Hinted at in CS151*
 - Given dynamic input, how does the output change over time?

Some examples of combinational circuits

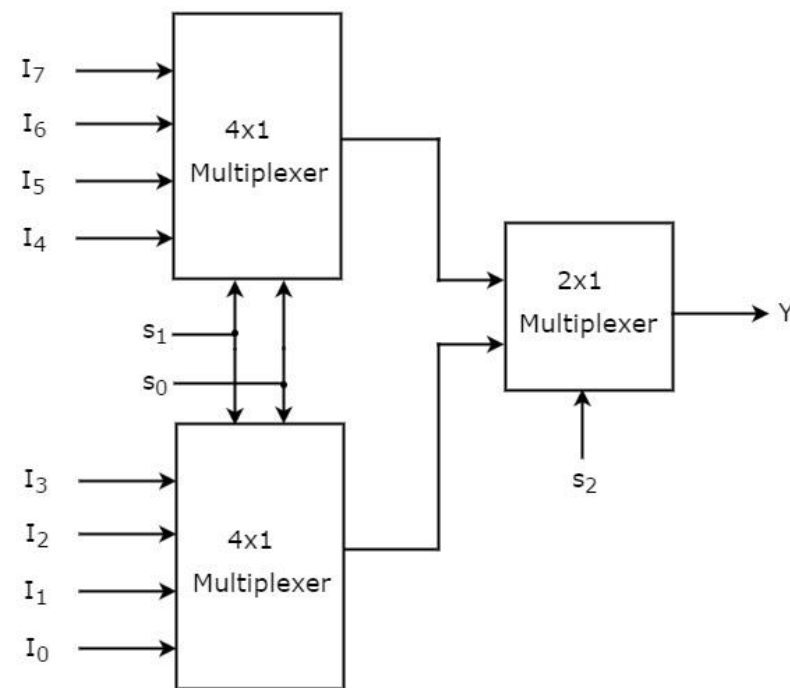
- ❑ Multiplexer selects one input signal (A/B) based on the control (S)
- ❑ Wider fan-in muxes can be built hierarchically



2-input mux



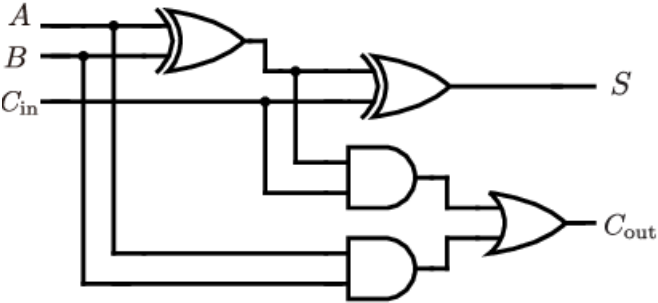
Source: H&P textbook



Hierarchical design of a
8x1 multiplexer

Some examples of combinational circuits

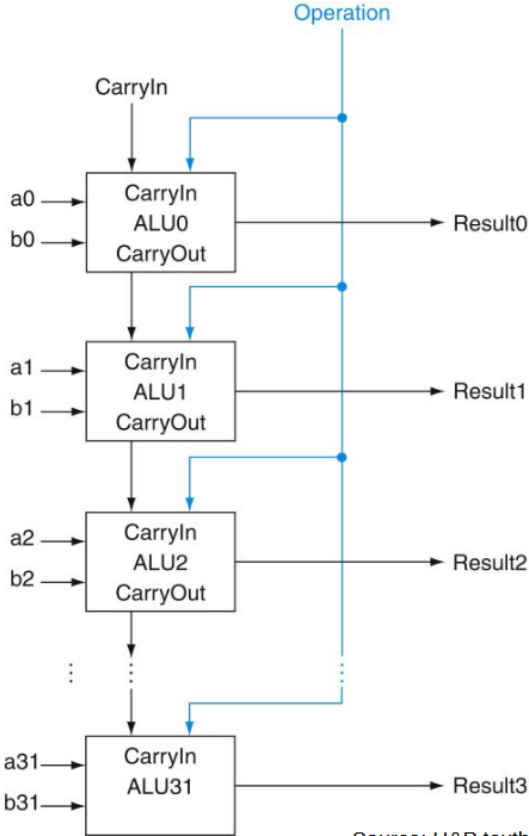
- Addition circuit chains together single-bit (“Full”) adders
 - 32 adders for 32-bit adder



Inputs			Outputs	
A	B	C _{in}	S	C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full adder

32-bit ripple carry adder

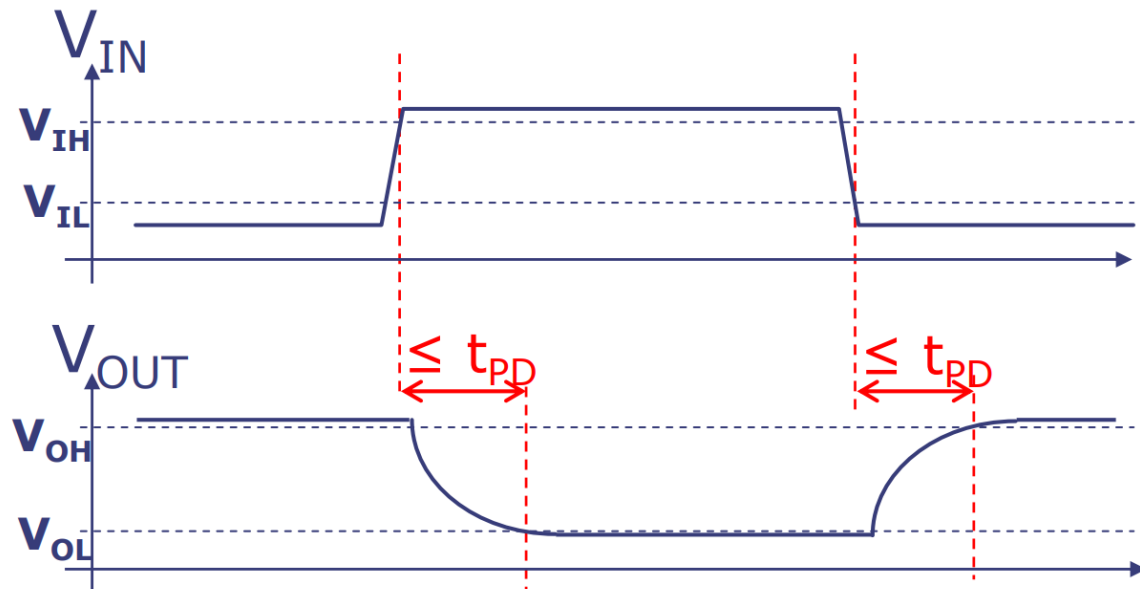


Source: H&P textbook

Timing specifications of combinational circuits

□ Propagation delay (t_{PD})

- An upper bound on the delay from valid inputs to valid outputs
- Restricts how fast input can be consumed
(Too fast input \rightarrow output cannot change in time, or undefined output)



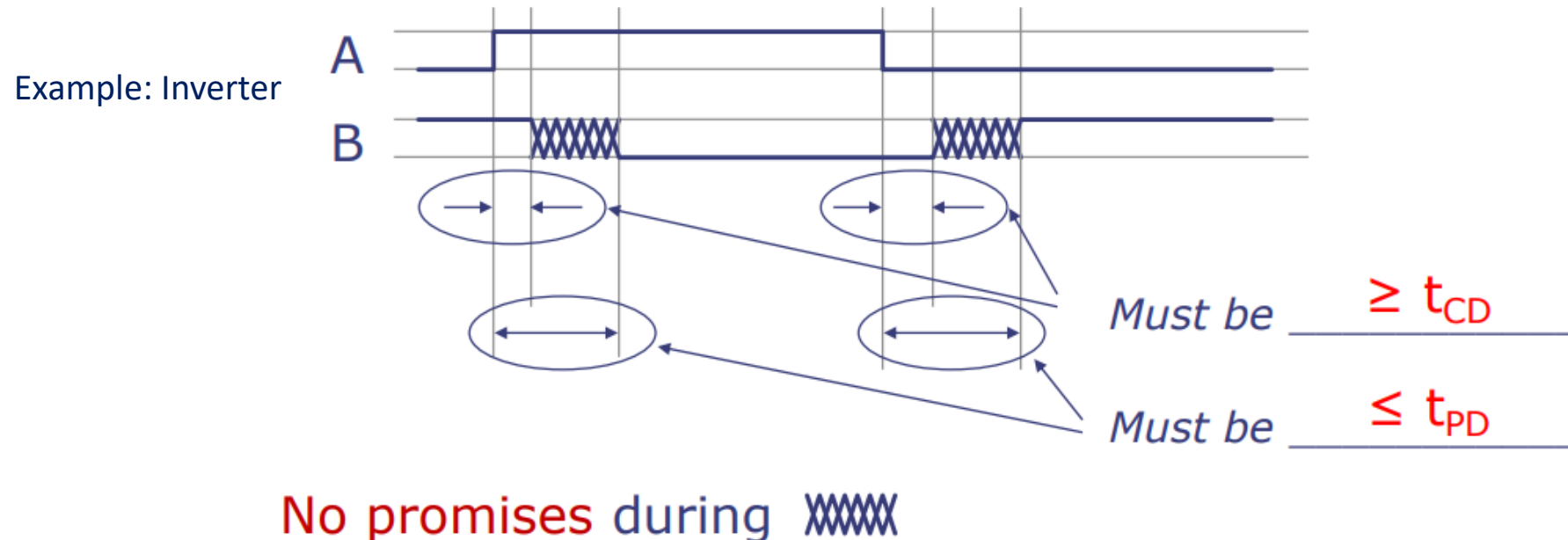
A good circuit has low t_{PD}
 \rightarrow Faster input
 \rightarrow Higher performance

How do we get low t_{PD} ?

Timing specifications of combinational circuits

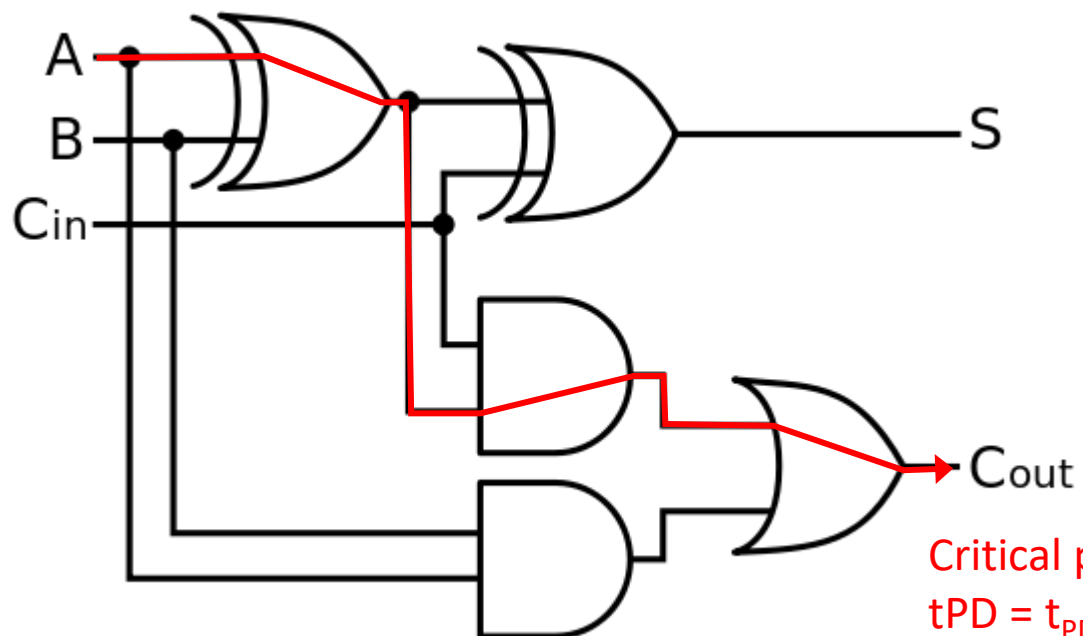
□ Contamination delay (t_{CD})

- A lower bound on the delay between input change to output starting to change
 - Does not mean output has stable value!
- Guarantees that output will not change within this timeframe regardless of what happens to input



Additive delay of combinational circuits

- ❑ A chain of logic components has additive delay
 - The “depth” of combinational circuits is important
- ❑ The “critical path” defines the overall propagation delay of a circuit



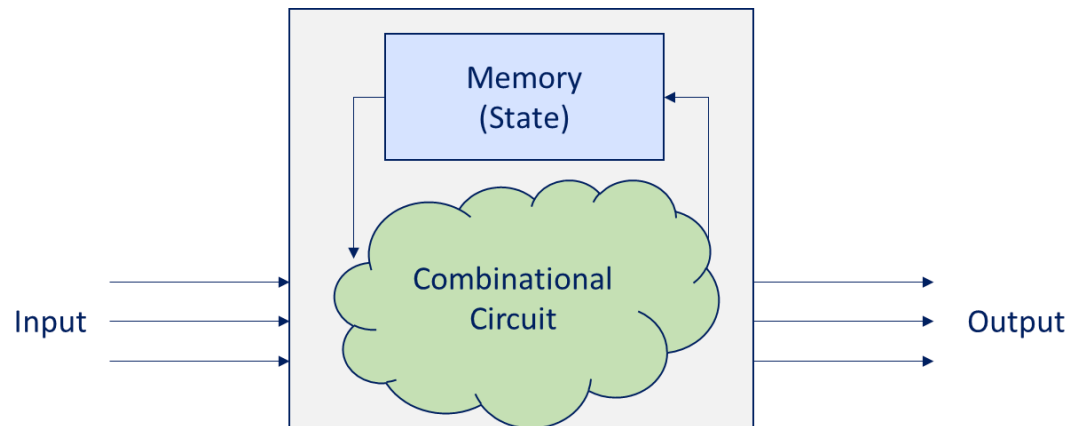
Long critical path
→ Slower input rate!

Critical path of three components
 $t_{PD} = t_{PD}(\text{xor2}) + t_{PD}(\text{and2}) + t_{PD}(\text{or2})$

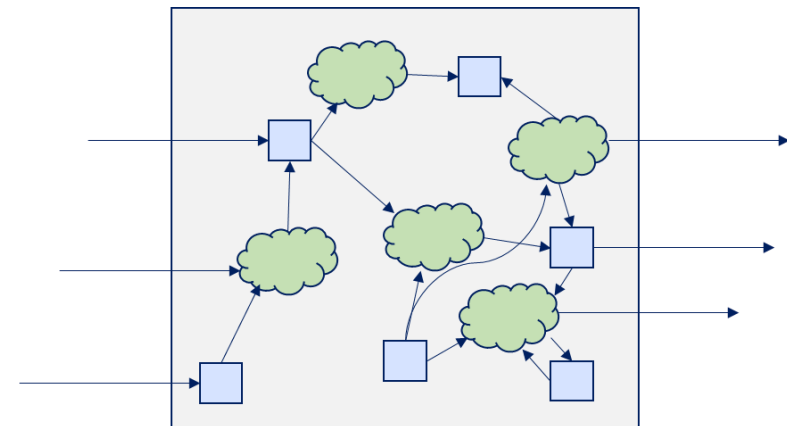
Example: A full adder

Sequential circuits

- ❑ Combinational circuits on their own are not very useful
- ❑ Sequential logic has memory (“state”)
 - State acts as input to internal combinational circuit
 - Subset of the combinational circuit output updates state



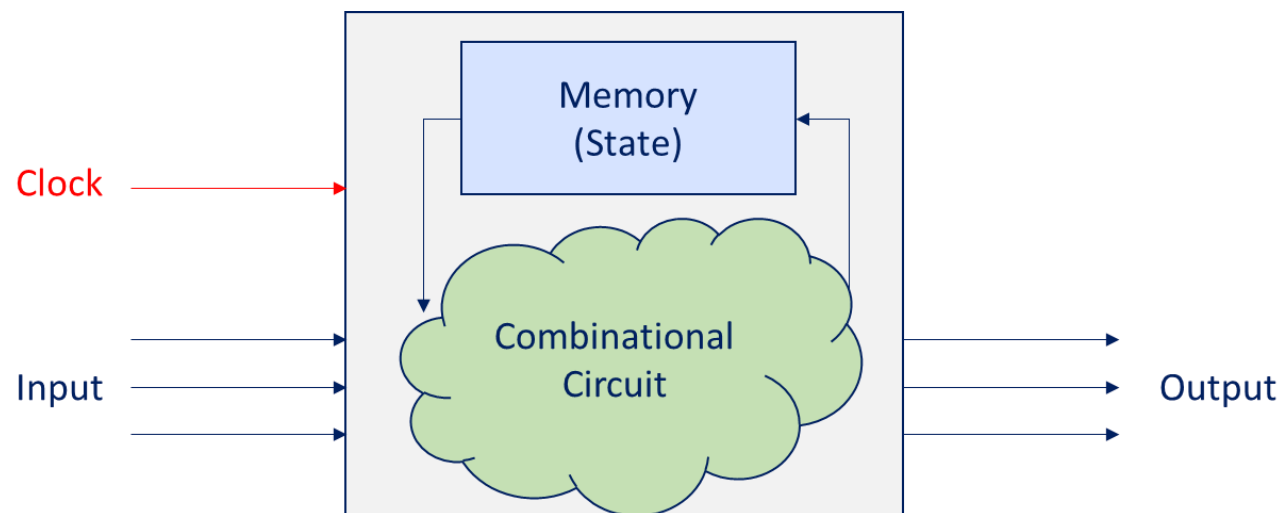
Abstract model of
Sequential circuits



Slightly more realistic
Sequential circuit

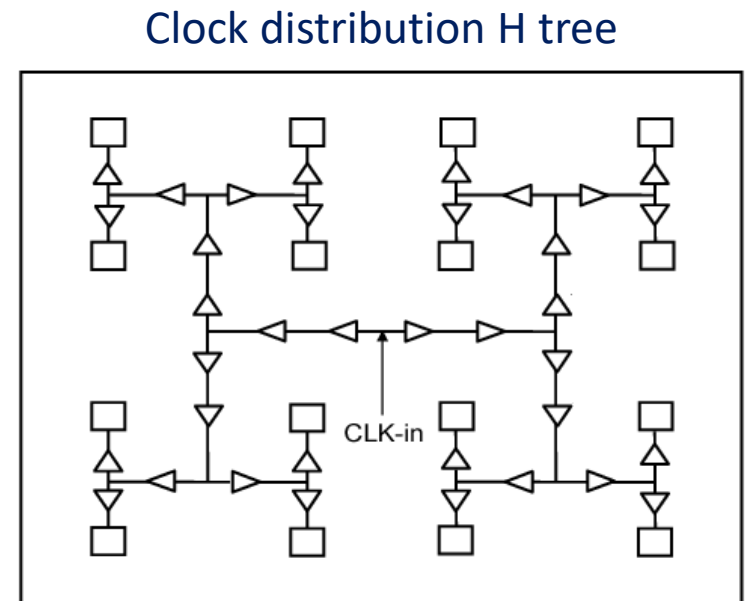
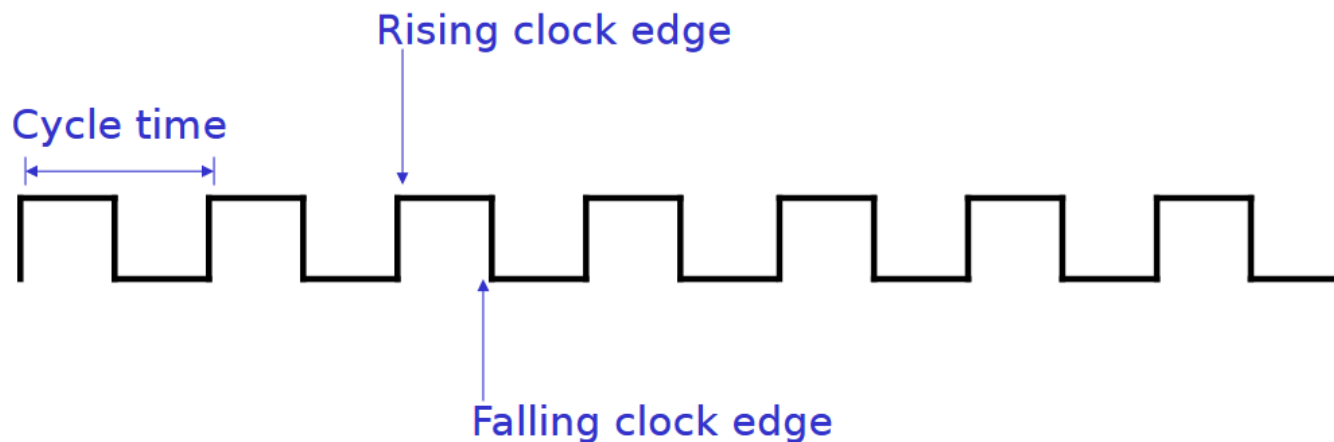
Synchronous sequential circuits

- ❑ “Synchronous”: all operations are aligned to a shared “clock” signal
 - Speed of the circuit determined by the delay of its longest critical path
 - For correct operation, all paths must be shorter than clock speed
 - Either simplify logic, or reduce clock speed!



A bit more about clocks

- All components of a synchronous circuit shares a common clock signal
 - Typically dynamic behavior starts at rising clock edge
 - Clocks propagated via special “clock tree” wires



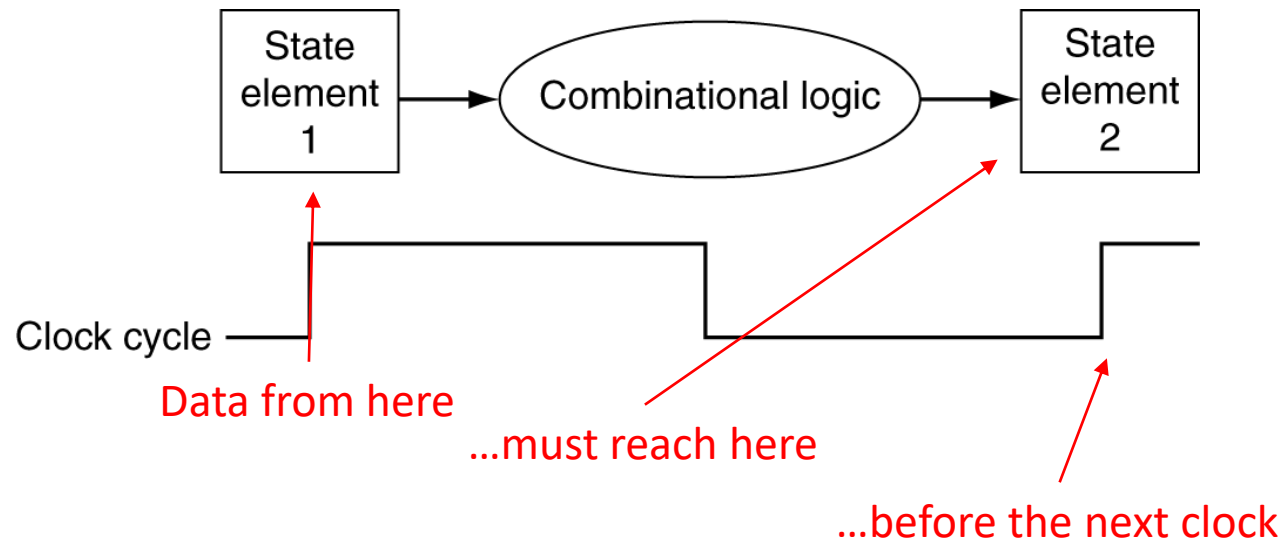
Timing constraints of state elements

- ❑ Synchronous state elements also add timing complexities
 - Beyond propagation delay and contamination delay
- ❑ Propagation delay (t_{PD}) of state elements
 - Rising edge of the clock to valid output from state element
- ❑ Contamination delay (t_{CD})
 - State element output does not change for t_{CD} after clock change
- ❑ Setup time (t_{SETUP})
 - It takes t_{SETUP} times for new input to register in state element
- ❑ Hold time (t_{HOLD})
 - Input to state element should hold correct data for t_{HOLD} after clock edge

Timing behavior of state elements

□ Meeting the setup time constraint

- “Processing must fit in clock cycle”
- After rising clock edge,
- $t_{pD}(\text{State element 1}) + t_{pD}(\text{Combinational logic}) + t_{\text{SETUP}}(\text{State element 2})$
- must be **smaller** than the clock period

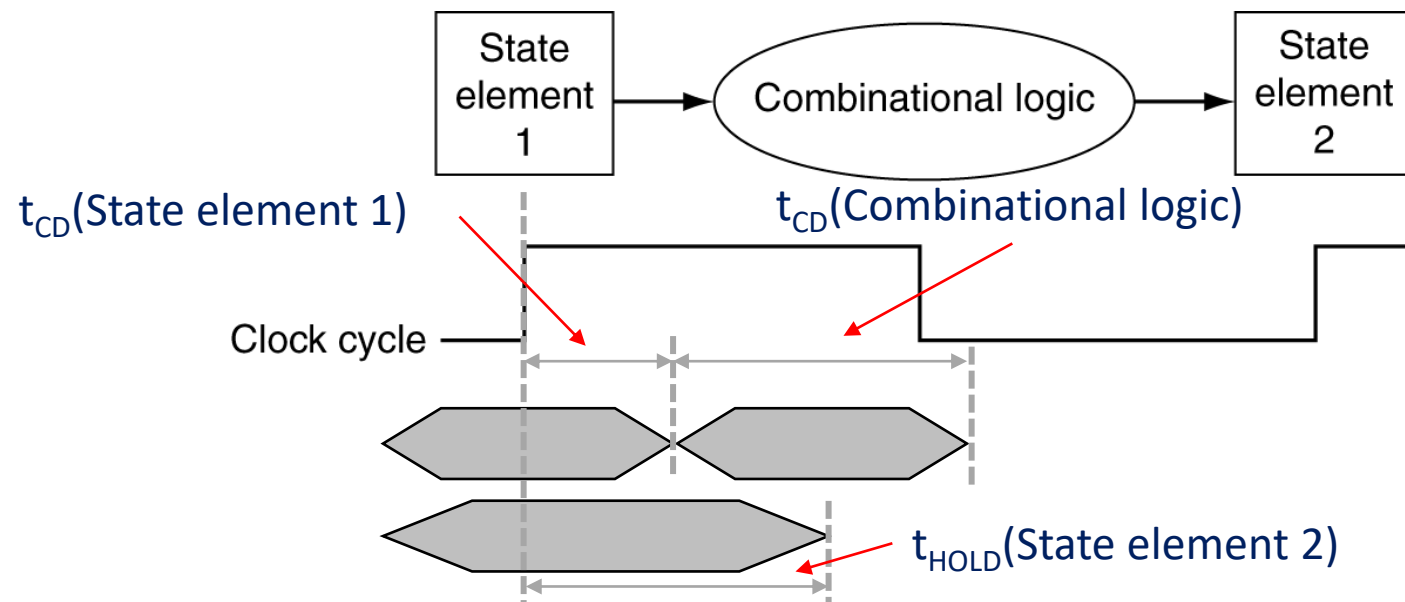


Otherwise, “timing violation”

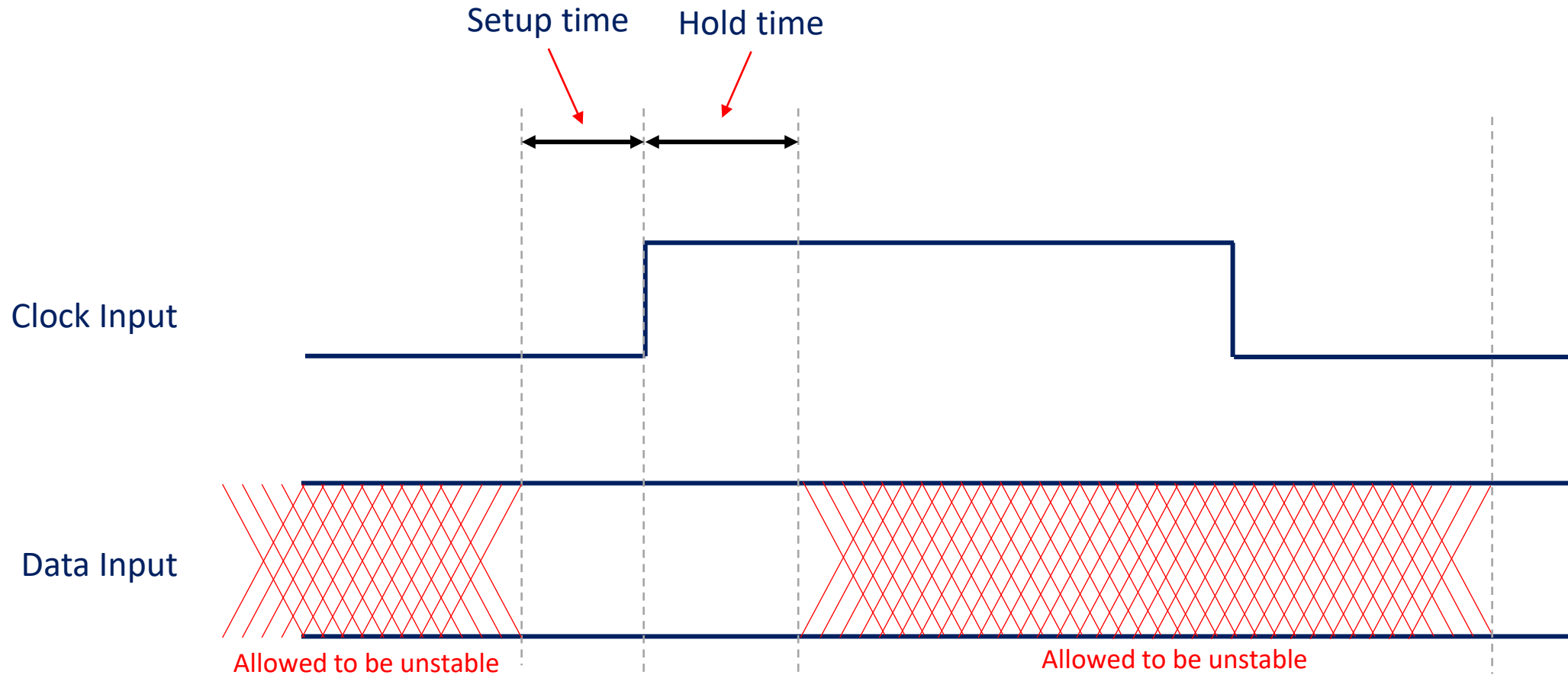
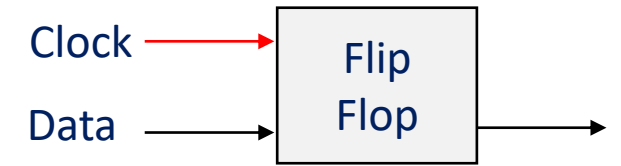
Timing behavior of state elements

□ Meeting the hold time constraint

- “Processing should not effect state too early”
- After rising clock edge,
- $t_{CD}(\text{State element 1}) + t_{CD}(\text{Combinational logic}) = \text{Guaranteed time output will not change}$
- must be **larger** than $t_{\text{HOLD}}(\text{State element 2})$



Setup and hold time window



If any constraint is violated, state may hold wrong data!

Real-world implications

- ❑ Constraints are met via Computer-Aided Design (CAD) tools
 - Cannot do by hand!
 - Given a high-level representation of function, CAD tools will try to create a physical circuit representation that meets all constraints

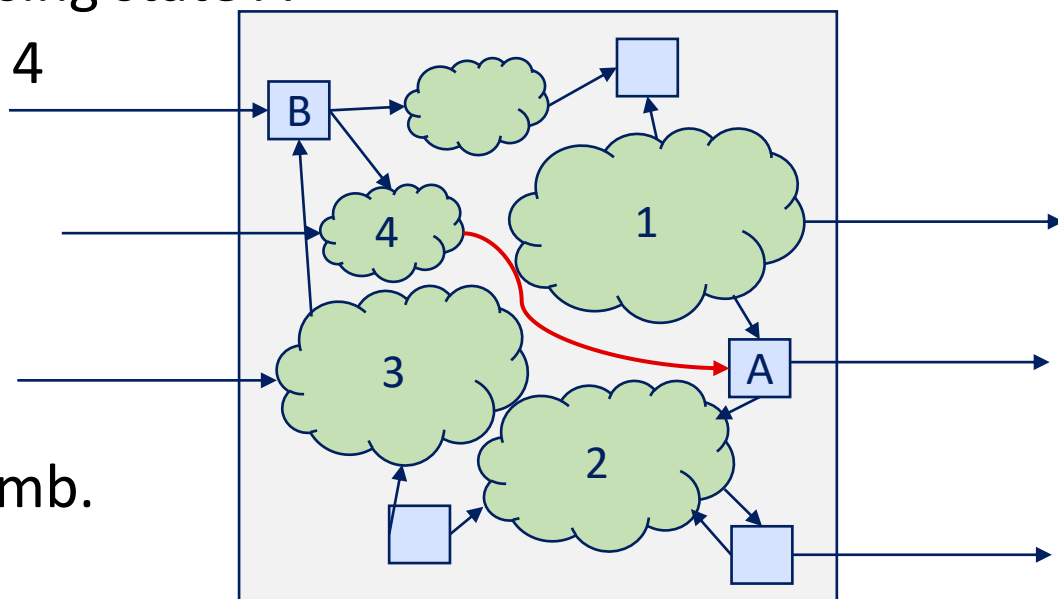
- ❑ Rule of thumb: Meeting **hold time** is typically not difficult
 - e.g., Adding a bunch of buffers can add enough t_{CD} (Sequential Circuit)

- ❑ Rule of thumb: Meeting **setup time** is often difficult
 - Somehow construct shorter critical paths, or
 - reduce clock speed (We want to avoid this!)

How do we create shorter critical paths for the same function?

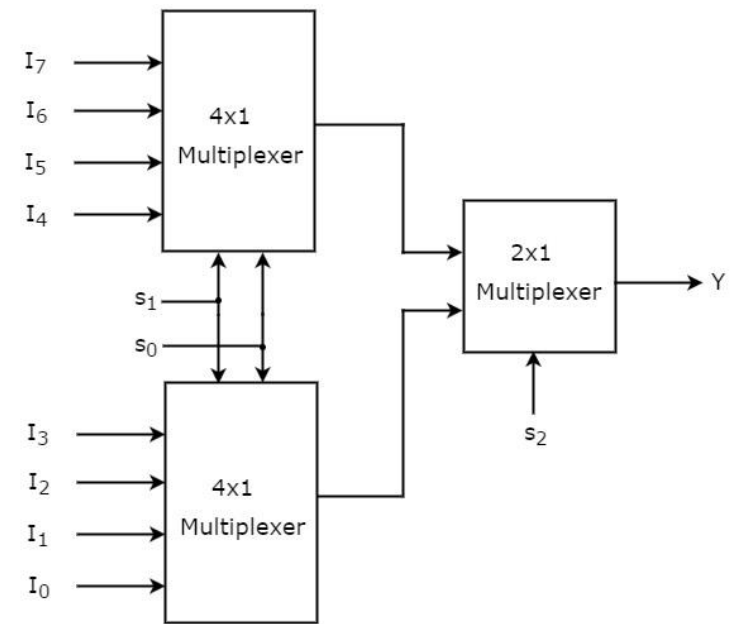
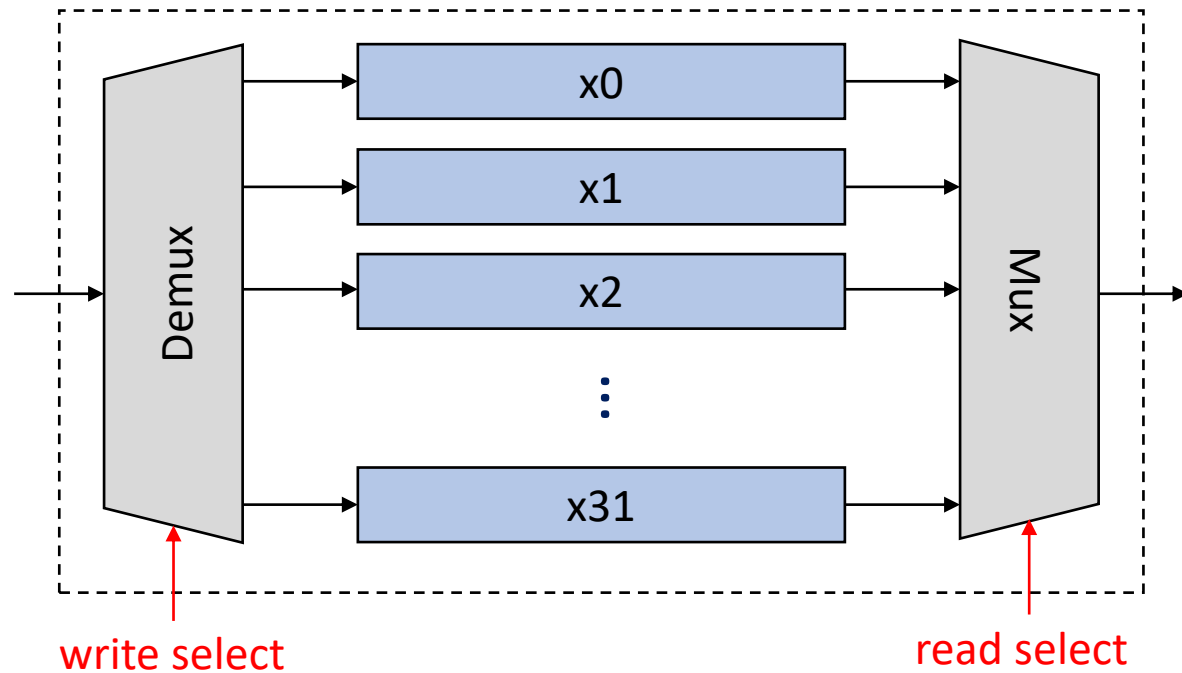
Simplified introduction to placement/routing

- ❑ Mapping state elements and combinational circuits to limited chip space
 - Also done via CAD tools
 - May add significant propagation delay to combinational circuits
- ❑ Example:
 - Complex combinational circuits 1 and 2 accessing state A
 - Spatial constraints push combinational circuit 4 far from state A
 - Path from B to A via 4 is now very long!
- ❑ Rule of thumb:
 - One comb. should not access too many state
 - One state should not be used by too many comb.



Looking back: Why are register files small?

- Why are register files 32-element? Why not 1024 or more?



Hierarchical design of a
8x1 multiplexer

Propagation delay increases with more registers! → Slower clock!

Real-world example

- ❑ Back in 2002 (When frequency scaling was going strong, but larger FETs)
 - Very high frequency (multi-GHz) meant:
 - ... setup time constraint could tolerate
 - ... up to 8 inverters in its critical path
 - Such stringent restrictions!

Can we even fit a 32-bit adder there? No!

Limit to reducing critical path → Limit to clock speed

Also, Dennard scaling ended, ending clock speed scaling in general

“Complex ISA can slow down the clock”

Why?

```
If (encoding[0] == True )  
    param1 = encoding[15:8];  
else  
    param1 = encoding[31:16];
```

Adds a MUX latency to critical path...

After Considering Constraint #1,#2

- ❑ The size of our processor may be limited...
- ❑ The clock speed may be limited...
 - How do we optimize the ISA and microarchitecture to achieve best clock speed?
 - Without using too much chip space?

- ❑ Can we do more work per clock cycle?

Constraints involved in processor design

Chip Area

Attainable Clock Speed



Instruction-Level Parallelism

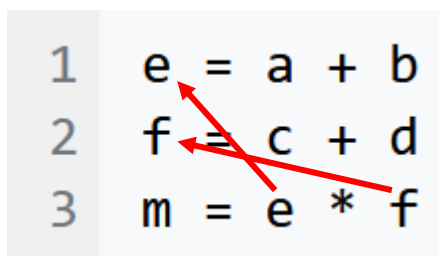
Amdahl's Law

Constraint #3: Instruction-Level Parallelism

- Given a sequence of instructions
 - Some instructions can be executed independently
 - If the processor has multiple execution units

https://en.wikipedia.org/wiki/Instruction-level_parallelism

```
1  e = a + b
2  f = c + d
3  m = e * f
```



Operations 1 and 2 are independent

Operation 3 depends on the results of 1 and 2

If our processor has two adders, 1 and 2 can be processed in parallel!

Note: ILP is about extracting parallelism from a SINGLE THREAD

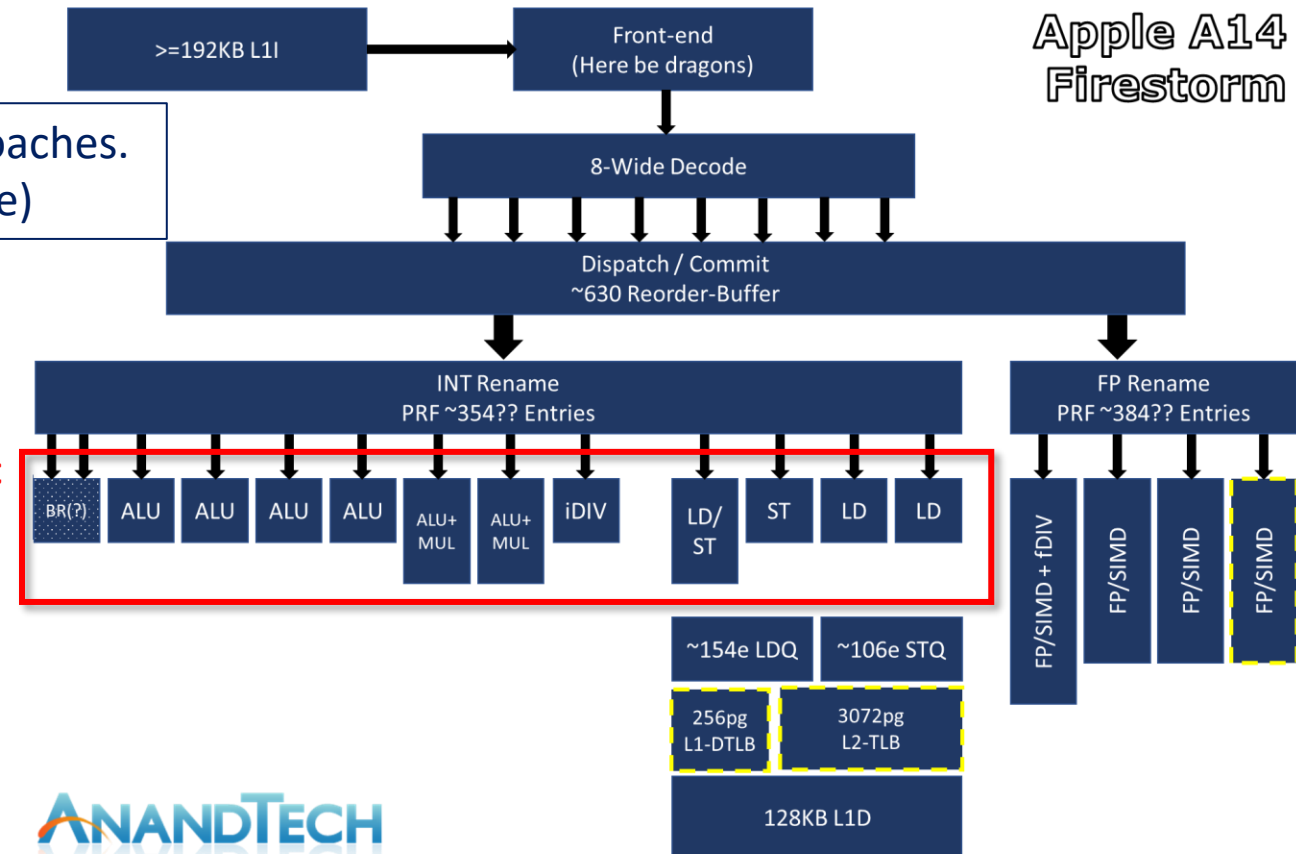
Microarchitectural Approaches to Exploit ILP

- Many approaches for discovering ILP and exploiting it
 - Pipelining, Superscalar, Out-of-Order, ...
 - We will talk about soon!

These are TRANSPARENT, MICROARCHITECTURAL approaches.
(Software/Assembly is written without being aware)

Many ALUs, etc
execute many instructions per clock cycle!

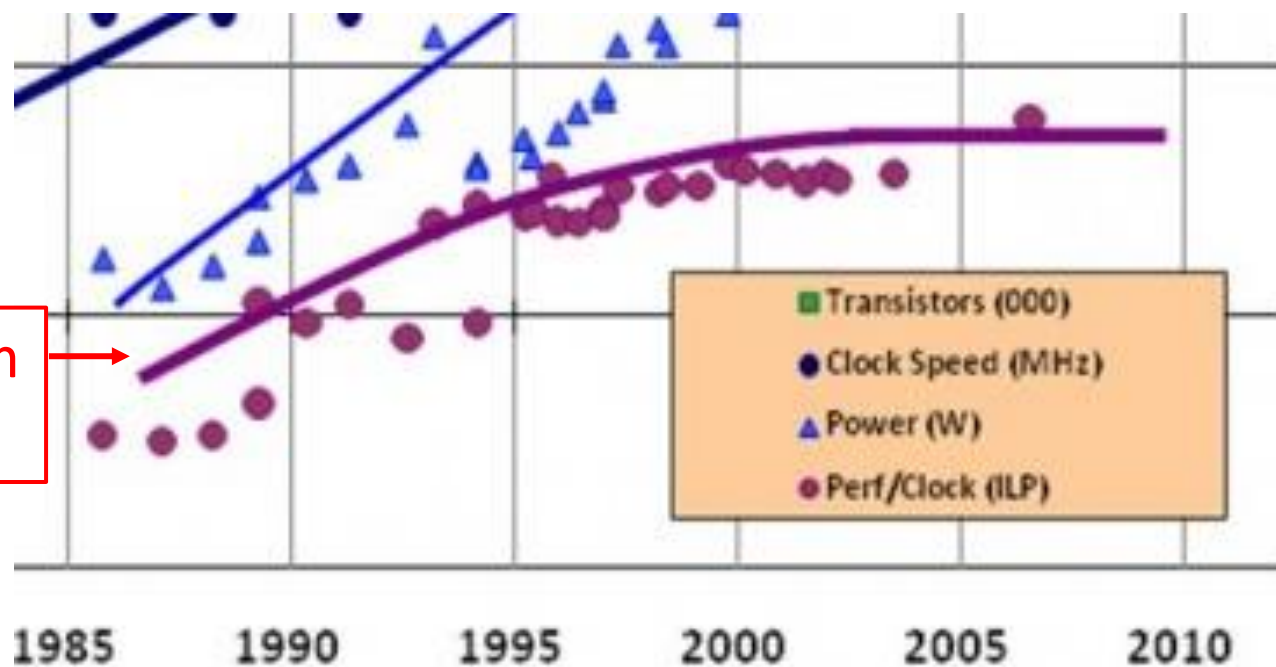
Can we keep adding ALUs forever?



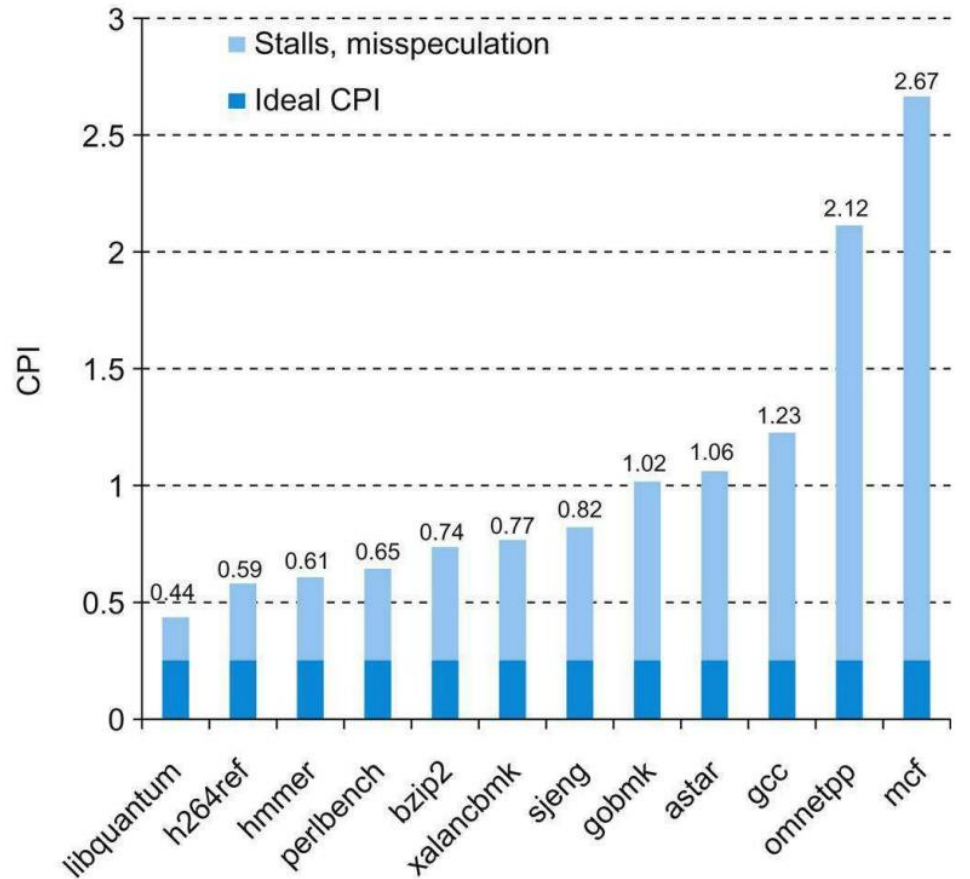
Constraint #3: Instruction-Level Parallelism

- ❑ No dramatic recent developments to exploit for ILP
- ❑ Later instructions, statistically, do depend on earlier ones

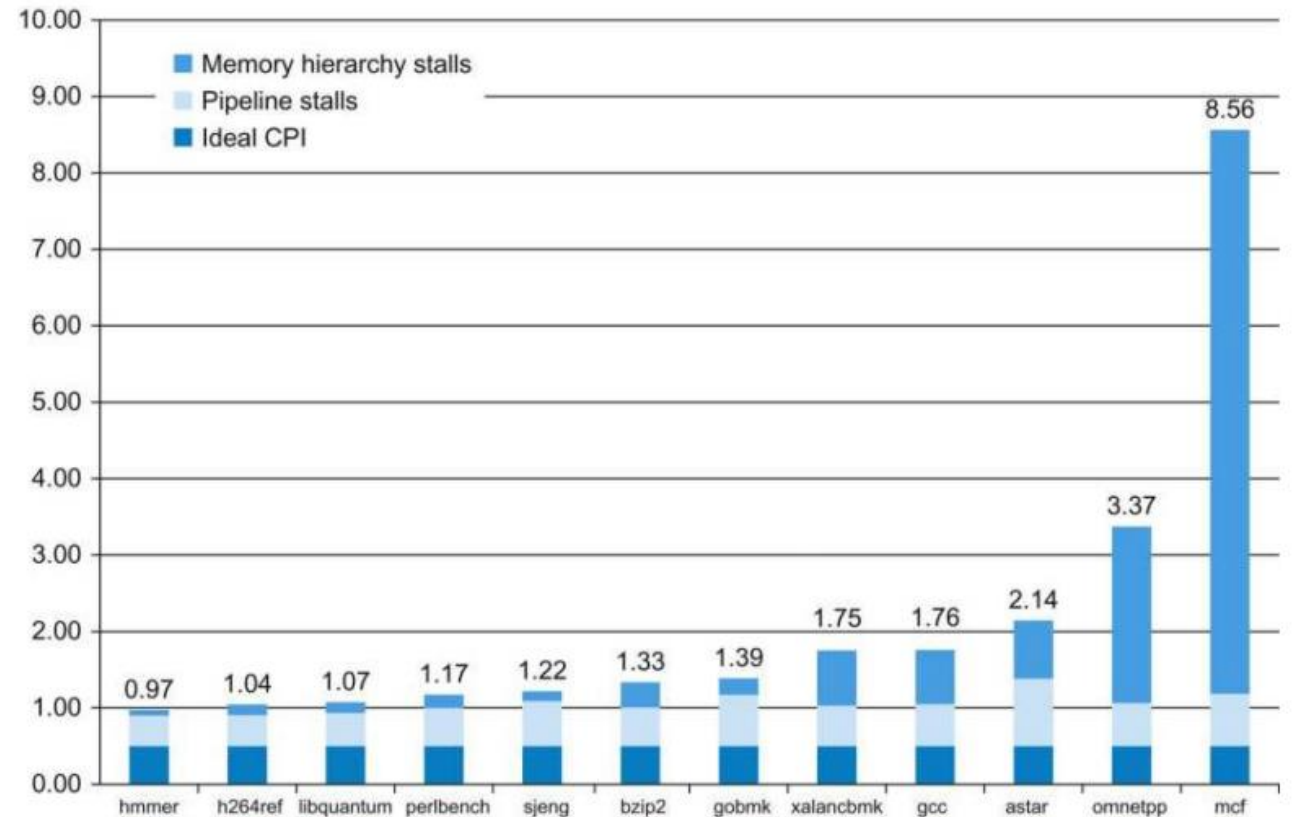
Instruction Level Parallelism shows diminishing returns



Real-world examples: Intel i7 and ARM Cortex-A53



CPI of Intel i7 920 on SPEC2006 Benchmarks



CPI of ARM Cortex-A53 on SPEC2006 Benchmarks

(High CPI is not ARM-inherent. Newer A78 has ideal 6 CPI)

After Considering Constraint #1,#2,#3

- The size of our processor may be limited...
- The clock speed may be limited...
- Work per clock cycle may be limited...
 - How much chip space do we want to invest in diminishing return ILP?

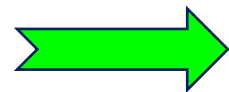
- How about if we had more cores?

Constraints involved in processor design

Chip Area

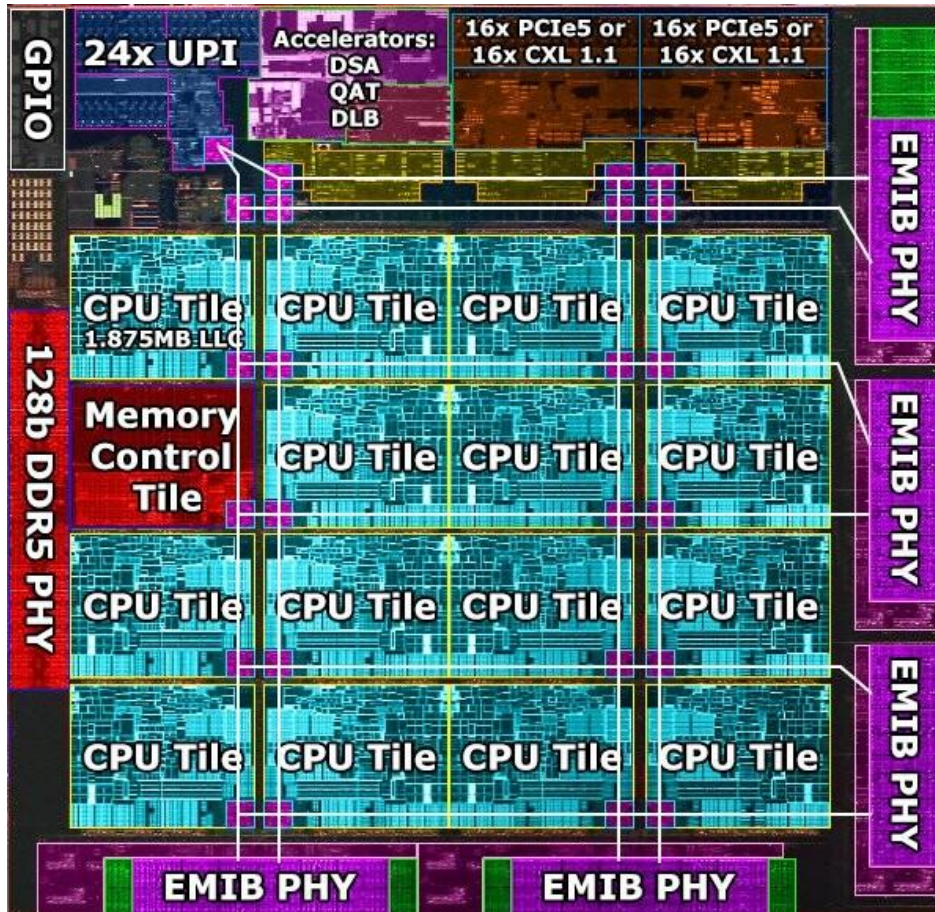
Attainable Clock Speed

Instruction-Level Parallelism



Amdahl's Law

Modern processors are often multicore



Modern machines run hundreds of threads

How about less investment in ILP,
and simply more cores?

Sure, if we don't care about the speed of any one task

Constraint #4: Amdahl's Law

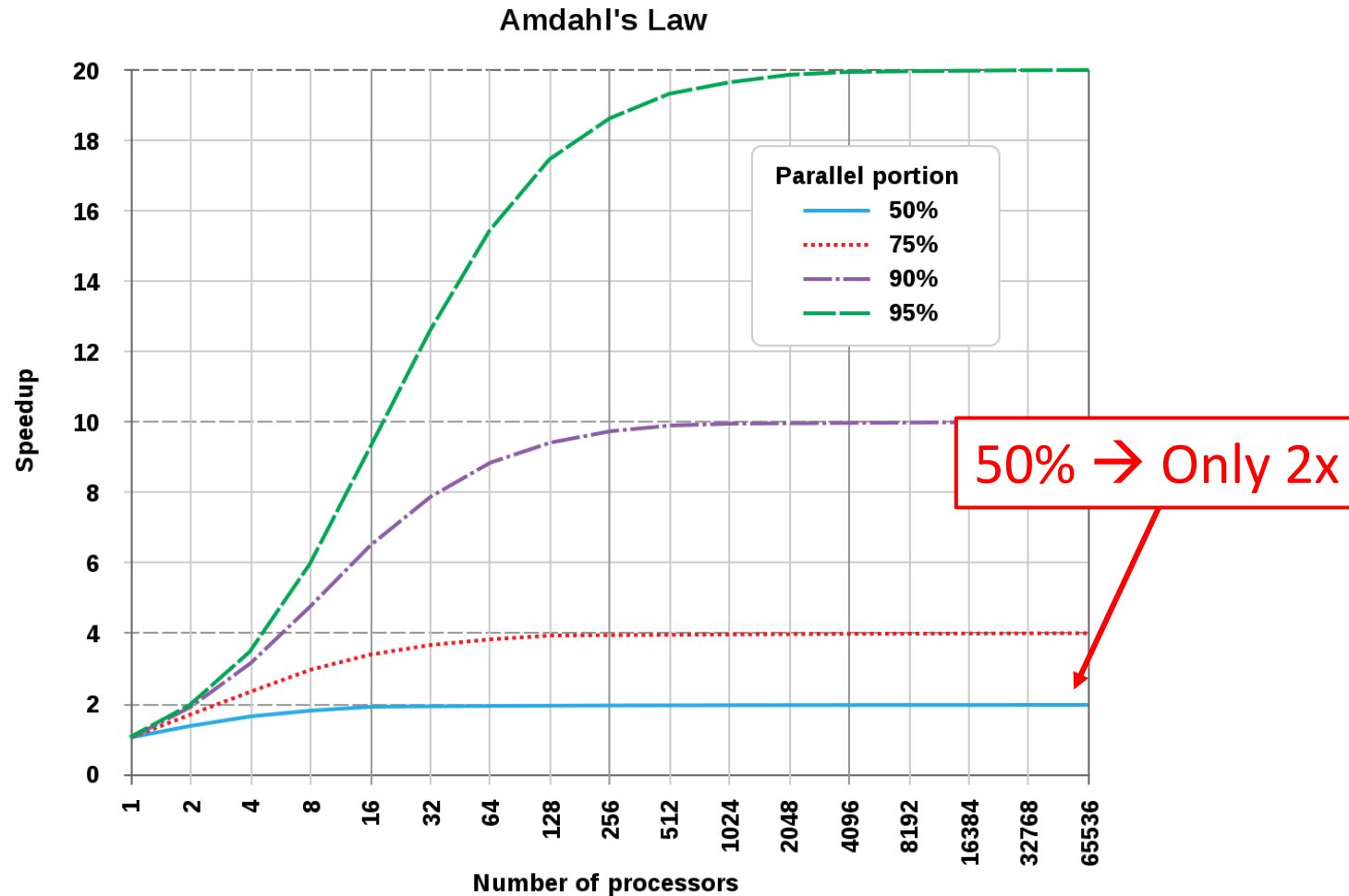
- Intuition: Given a task, some parts of it are inherently not parallelizable
 - Dependencies...

$$\text{Speedup} = \frac{1}{\frac{F_{parallel}}{n} + (1 - F_{parallel})}$$

Parallelizable fraction

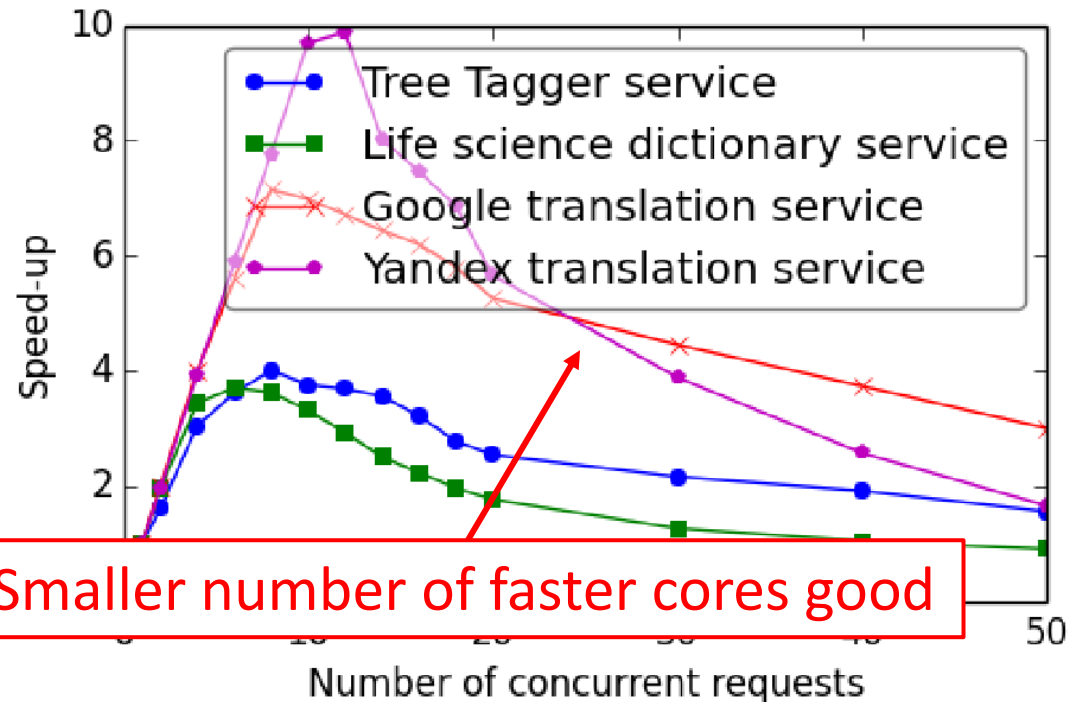
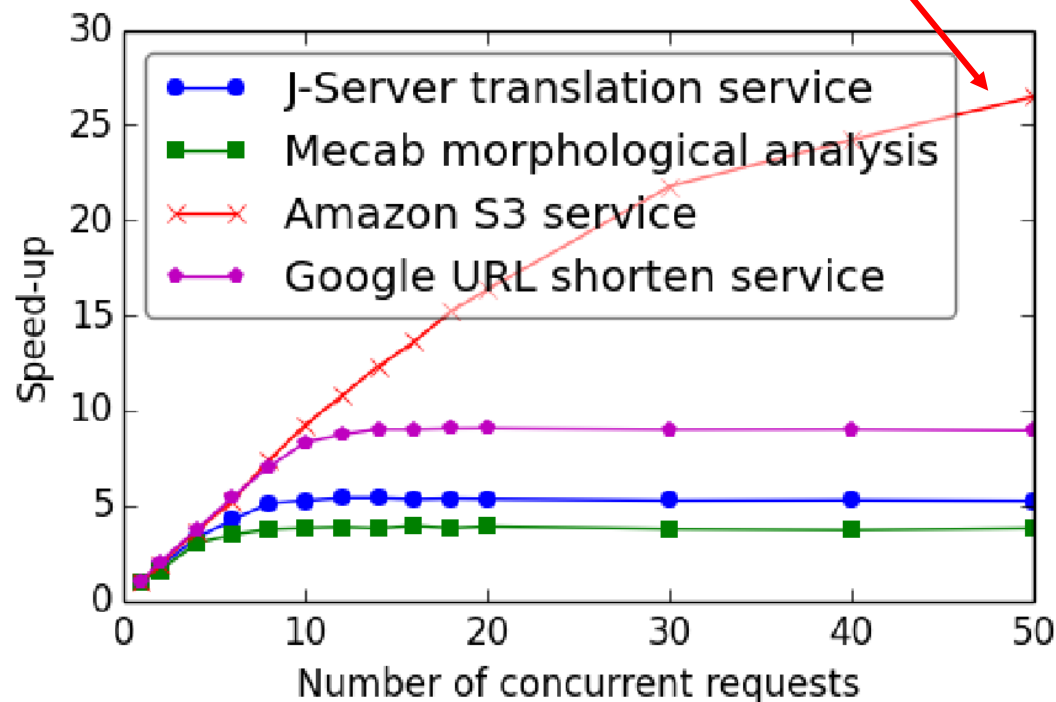
Non-parallelizable fraction

Constraint #4: Amdahl's Law



Real-World is Often Worse

Larger number of slower cores good



Smaller number of faster cores good

Due to other overheads including orchestrating parallel operations

After Considering Constraint #1,#2,#3,#4

- ❑ The size of our processor may be limited...
- ❑ The clock speed may be limited...
- ❑ Work per clock cycle may be limited...
- ❑ Benefits of more cores may be limited...
 - How do we balance core count vs ILP, given chip space restrictions?

No resource can scale forever!
Need to balance... How?

Application, benchmark-driven!