Systemarchitektur
Sommersemester 2021 – Recap

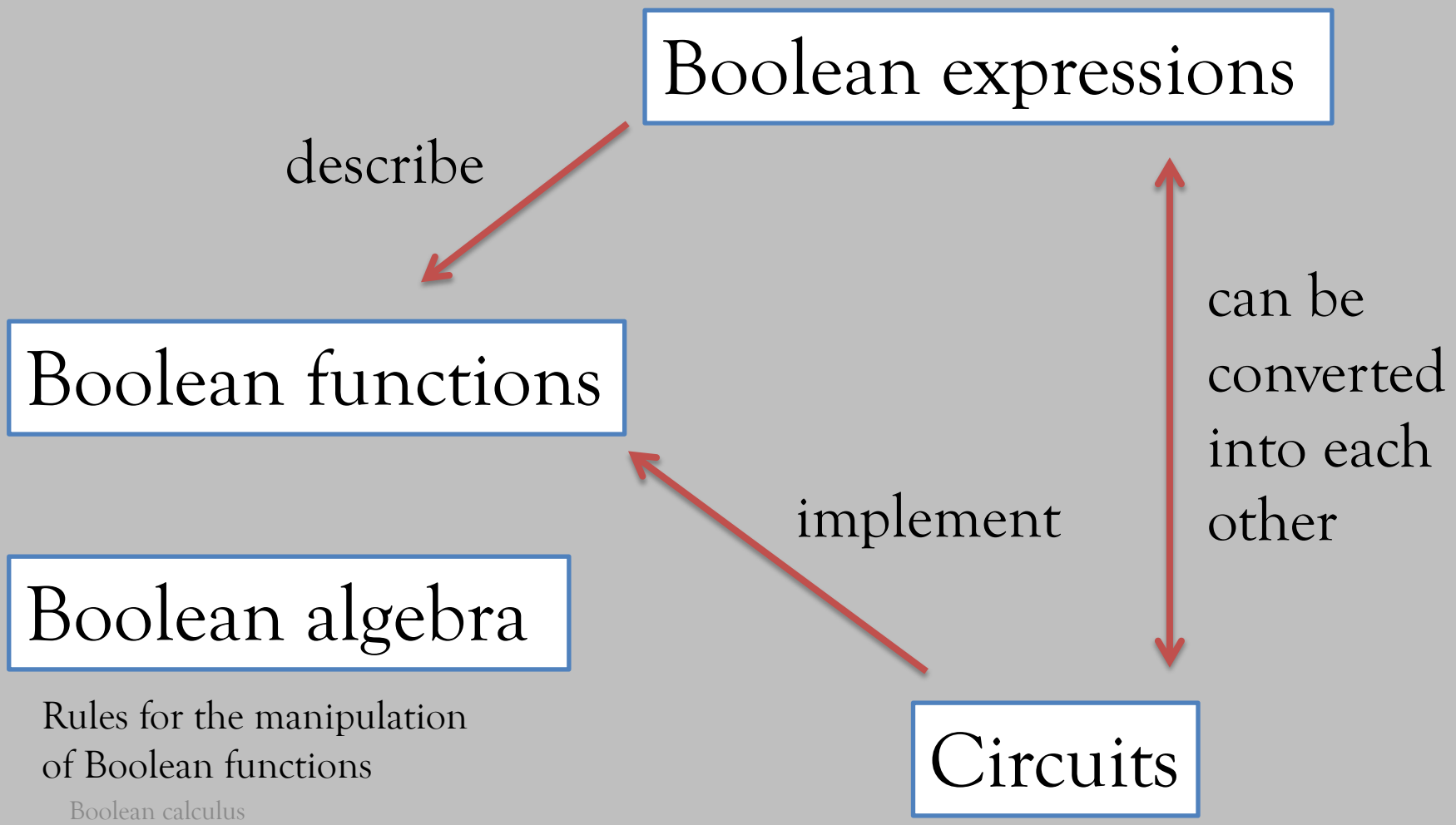
Jan Reineke
Universität des Saarlandes

Reminder - Exam Registration

- Final Exam: July 30, 2021, 10:00-12:00
- Registration in LSF: July 23, 2021

- Those who **cannot** register in LSF:
  - We will open a registration in the CMS on Saturday
  - Register by Monday, July 26

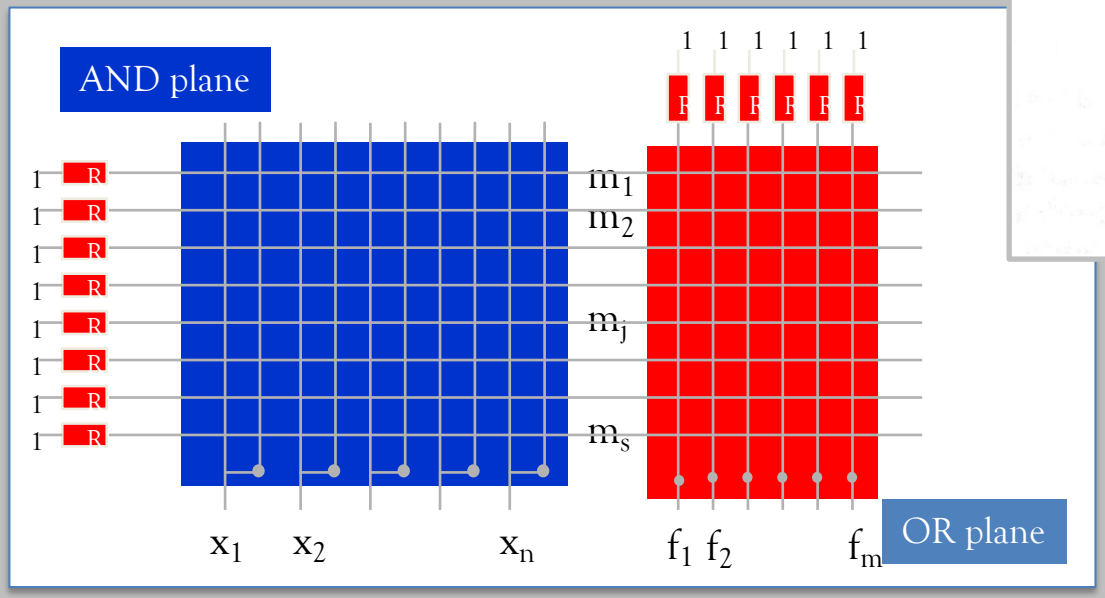
1. Boolean Calculus

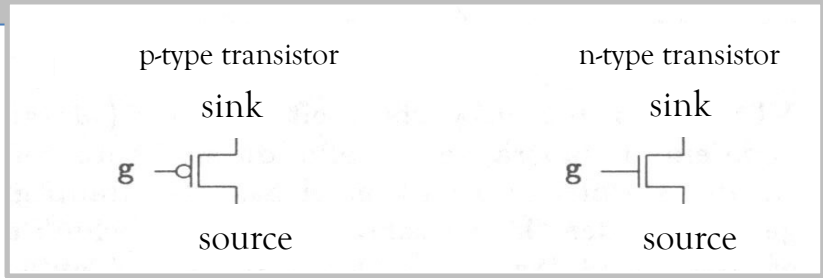


1. Boolean Calculus – Key Items

- Partial and total Boolean functions
- Truth tables, On-Set, Off-Set
- Boolean algebra, axioms, laws, duality principle
- Boolean expressions, syntax, semantics
- Literals, monomials, polynomials
- (Canonical) disjunctive/conjunctive normal form

2. PLAs and Logic Synthesis





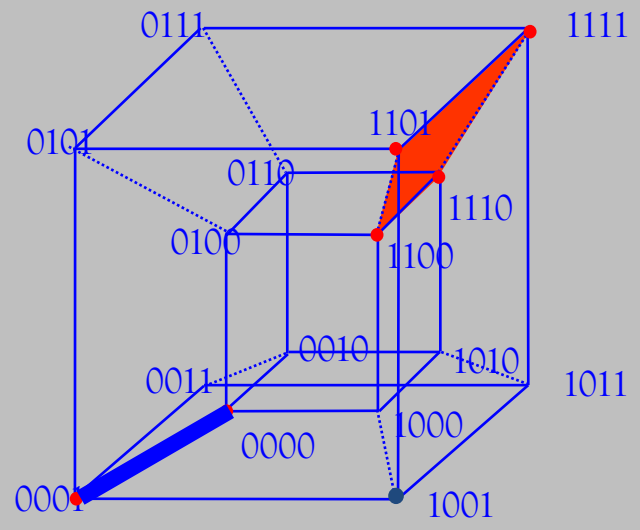
Example:

$f(x_1,x_2,x_3,x_4)$

$= x_1x_2$

$+ x_1'x_2'x_3'$

$+ x_1x_2'x_3'x_4$



3. PLAs and Logic Synthesis – Key Items

- n-type and p-type transistors
- Programmable Logic Arrays (PLAs)
- Implementation of monomials and polynomials in PLAs
- Cost of monomials and polynomials
- Two-level logic minimization, and the corresponding covering problem on the hypercube

2. Implicants and Prime Implicants

An **implicant of f** is a monomial q with $\psi(q) \leq f$.
A **prime implicant of f** is a maximal implicant q of f.

Theorem (Quine):
Every minimal polynomial p of a Boolean function f consists only of prime implicants of f.

Theorem (Implicants):

A monomial m is an implicant of f if and only if, either

- m is a minterm of f, or
- m·x and m·x' are implicants of f
  for a variable x that does not occur in m.

Thus: $m \in \text{Implicant}(f) \Leftrightarrow$
$[m \in \text{Minterm}(f)] \vee [m \cdot x, m \cdot x' \in \text{Implicant}(f)]$

2. Implicants and Prime Implicants – Key Items

- Implicants and prime implicants
- Minimal polynomial
- Theorem of Quine
- Characterization of implicants

3. Quine/McCluskey Algorithm

Quine-Prime-Implicants(f: B^n -> B)

L_0 := Minterm(f)

$i := 1$

$Prime(f) := \emptyset$

while ($L_{i-1} \neq \emptyset$) and ($i \leq n$)

$L_i := \{m \mid |m|=n-i,\ m \cdot x \text{ and } m \cdot x' \text{ are in } L_{i-1} \text{ for some } x\}$

$P_i := \{m \mid m \in L_{i-1} \text{ and } m \text{ is not covered by any } m' \in L_i\}$

$Prime(f) := Prime(f) \cup P_i$

$i := i+1$

return $Prime(f) \cup L_{i-1}$

Quine's algorithm

McCluskey's Improvement

Compare only those monomials

- that contain the **same variables**, and
- whose number of **positive literals differs by one.**

3. Quine/McCluskey Algorithm

Matrix-covering problem

1. Reduction Rule:
Remove from the prime implicant table **PIT(f)** all essential prime implicants and all minterms that are covered by these prime implicants.

2. Reduction Rule:
Remove all minterms from the prime implicant table **PIT(f)** that dominate another minterm in **PIT(f)**.

3. Reduction Rule
Remove all prime implicants from the prime implicant table PIT(f) that are dominated by other prime implicants that are not more expensive.

3. Quine/McCluskey Algorithm – Key Items

- Quine's algorithm
- McCluskey's improvement
- Correctness and complexity of the Quine-McCluskey algorithm
- The matrix covering problem
- Three reduction rules
  - Essential prime implicants
  - Column domination
  - Row domination
- Cyclic covering problems, Petrick's method

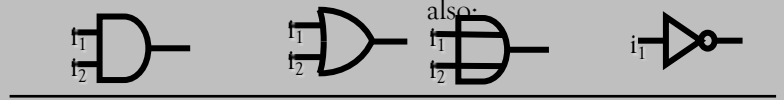
4. Combinatorial Circuits

Logic gates

i_2	i_1	AND_2
0	0	0
0	1	0
1	0	0
1	1	1

i_2	i_1	OR_2
0	0	0
0	1	1
1	0	1
1	1	1

i	NOT
0	1
1	0



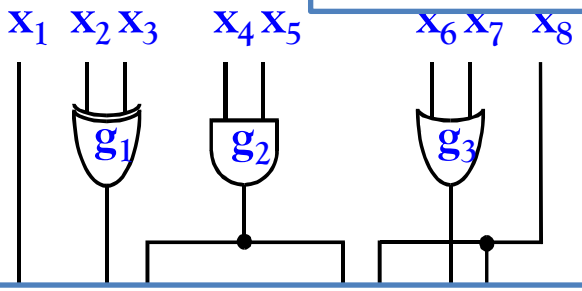
i_2	i_1	$NAND_2$
0	0	1
0	1	1
1	0	1

i_2	i_1	NOR_2
0	0	1
0	1	0
1	0	0

i_2	i_1	XOR_2
0	0	0
0	1	1
1	0	1

Circuits and their Formalization

"acyclic graph"



The **hardware cost** C(C) of a circuit C is its number of gates $|I| = |V \setminus (\{0, 1\} \cup (x_1, \ldots, x_n))|$.

The **depth** depth(C) of a circuit C is the **maximal number of gates on a path** from an arbitrary input x_i to an arbitrary output y_j of C.

4. Combinatorial Circuits – Key Items

- Logic gates, cell library
- Circuits
- Semantics of circuits
- Concrete and symbolic simulation
- Cost and depth of circuits
- Hierarchical circuits
- Circuits vs Boolean functions
- Implementation or associative operations

5. Number Representations

Questions:

1. How to represent *natural numbers*?
2. How to represent *integers*?
   Challenge: negative numbers
3. How to represent *rational numbers*?
4. How to represent *very large* and very *small numbers*?

} fixed-point numbers

} floating-point numbers

Binary numbers: $\langle d_n d_{n-1} \ldots d_0 \rangle := \sum_{i=0,\ldots,n} d_i \cdot 2^i$

Two's complement: $[d_n d_{n-1} \ldots d_0]_2 := \sum_{i=0,\ldots,n-1} d_i \cdot 2^i - d_n \cdot 2^n$

Fixed-point numbers: $[d_n d_{n-1} \ldots d_0, d_{-1} \ldots d_{-k}]_2 := \sum_{i=-k,\ldots,n-1} d_i \cdot 2^i - d_n \cdot 2^n$

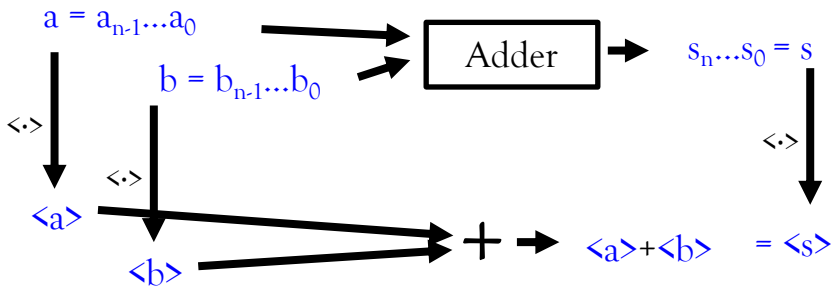
Floating-point numbers: sign, exponent, mantissa

5. Number Representations – Key Items

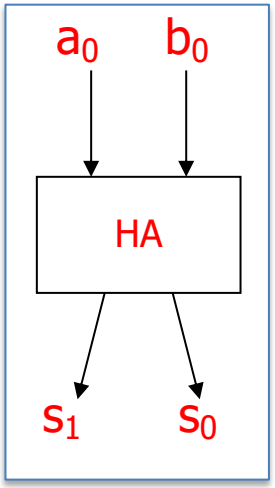
- Numerals (digits), binary, decimal, hexadecimal
- Positional numeral system
- Natural numbers
- Signed-magnitude representation, One's complement, Two's complement
- Fixed-point numbers
- Floating-point numbers

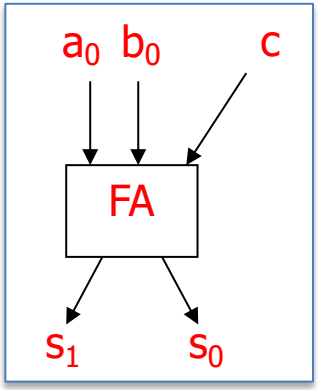
6. Arithmetic Circuits: Adders

Definition of an Adder

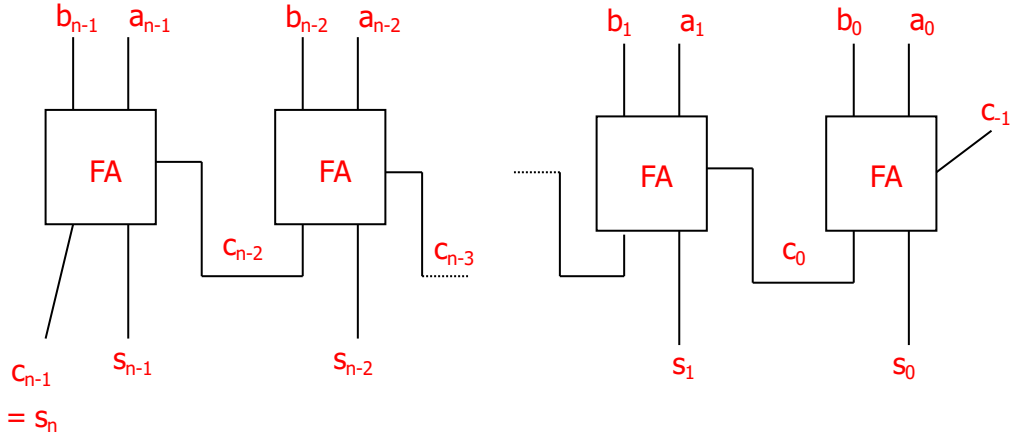


Half and Full adders





Ripple-carry Adders



$C(RC_n) = n \cdot C(FA) = 5n$

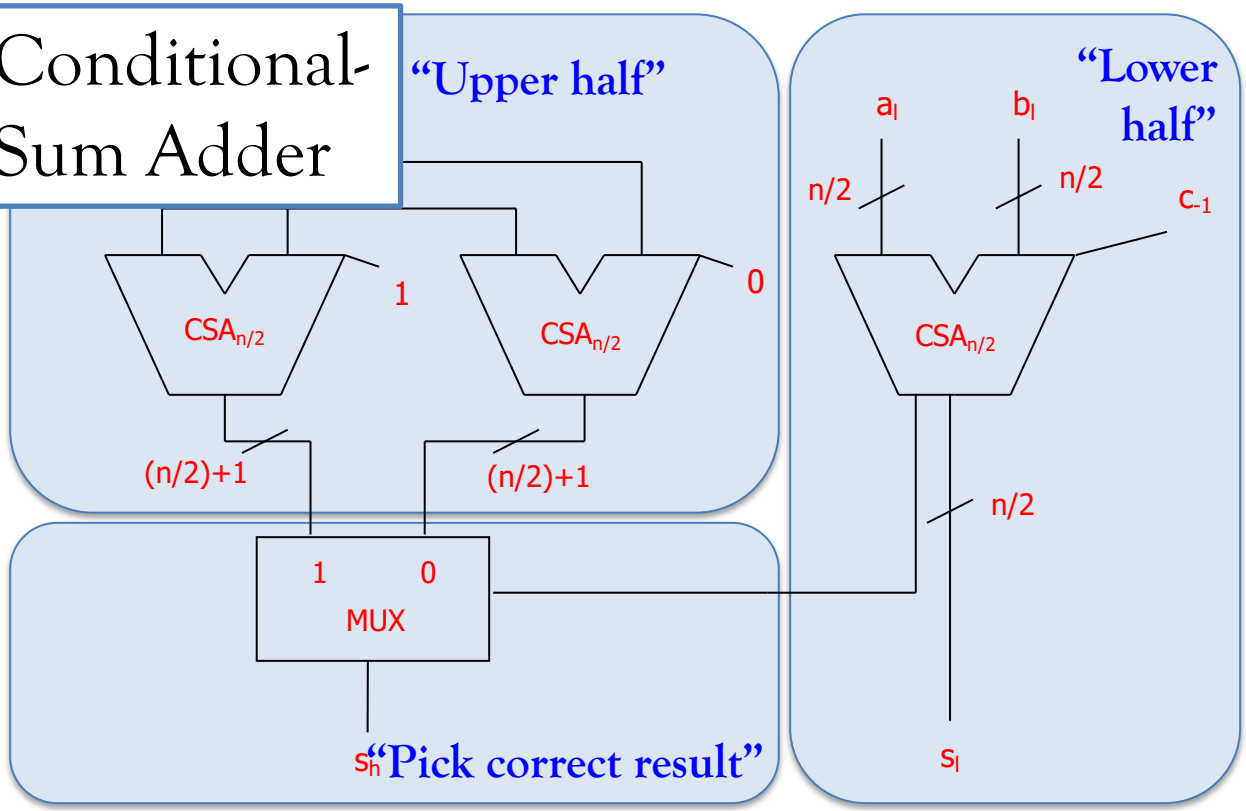
$depth(RC_n) = 3 + 2(n-1)$

6. Arithmetic Circuits: Adders

Lower bounds for adders!

$C(+_n) \geq 2n, \quad \text{depth}(+_n) \geq \log(n) + 1$

Conditional-Sum Adder



$\text{depth}(CSA_n) = 3 \log_2 n + 3$

$C(CSA_n) = 10n^{\log 3} - 3n - 2$

6. Arithmetic Circuits: Adders

Let M be a set and $\circ : M \times M \to M$ an **associative** operation.
The **parallel prefix sum** $PP^n: M^n \to M^n$ is defined as follows:

$$PP^n (x_{n-1}, \ldots, x_0) = (x_{n-1} \circ x_{n-2} \ldots \circ x_0, \ldots, x_1 \circ x_0, x_0)$$

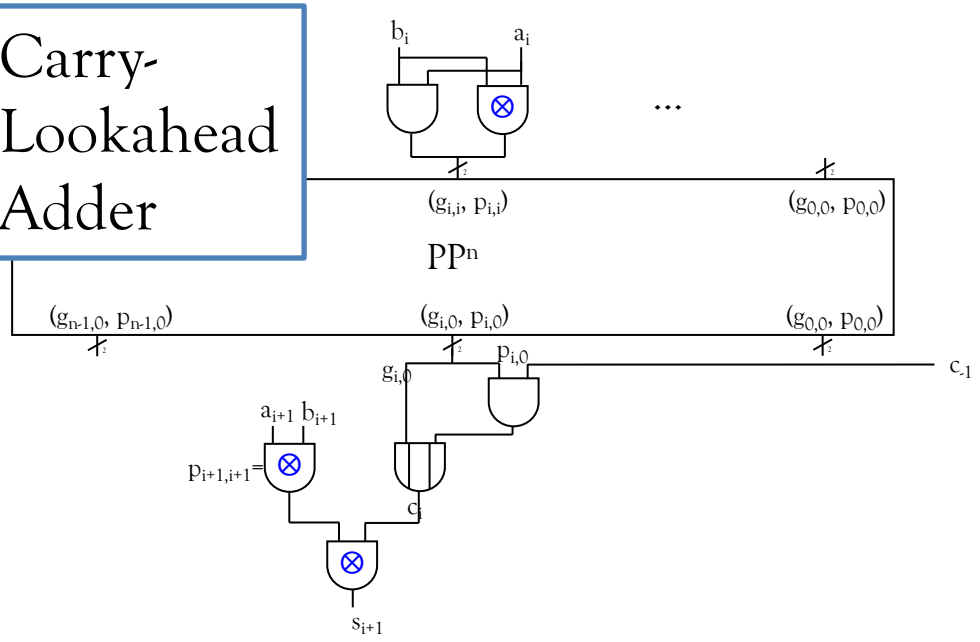
$$\text{depth}(PP^n) < (2 \cdot \log_2 n) \cdot \text{depth}(\circ)$$

$$C(PP^n) < 2n \cdot C(\circ)$$

Generated carry $g_{j,i}$ from i to j:
$c_j = 1$ *independently* of c_{i-1}.
Propagated carry $p_{j,i}$ from i to j:
$c_j = 1$ *if and only if* also $c_{i-1} = 1$

Generated and propagated carries can be captured as parallel prefixes of associative operator.

Carry-Lookahead Adder



6. Arithmetic Circuits: Adders – Key Items

- Definition of adder
- Half adder, Full adder
- Ripple-carry adder, correctness, cost, depth
- Recursive constructions, inductive proofs
- Incrementer, Multiplexer
- Lower bounds on cost and depth
- Conditional-sum adder, divide-and-conquer
- Addition in two's complement

6. Arithmetic Circuits: Adders – Key Items

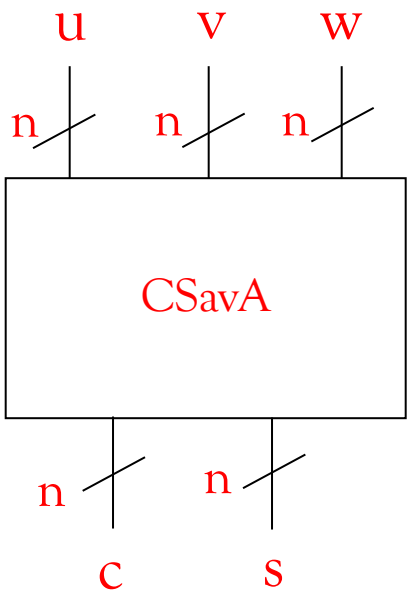
- Parallel prefix operation and circuit
- Generated and propagated carries
- Carry-lookahead adder

6. Arithmetic Circuits

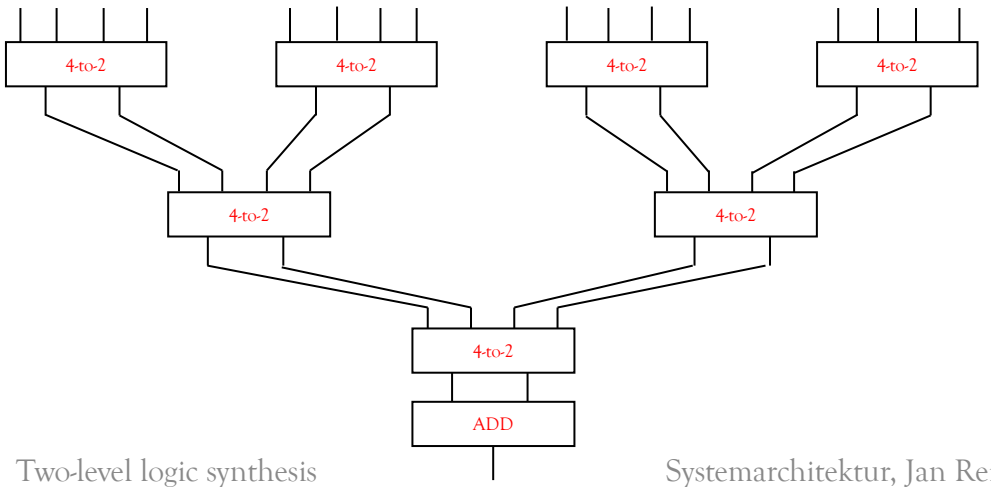
Multiplication matrix

$$\begin{pmatrix} pp_0 \\ pp_1 \\ \vdots \\ pp_{n-1} \end{pmatrix} = \begin{pmatrix} 0 & 0 & \dots & 0 & 0 & a_{n-1}b_0 & a_{n-2}b_0 & \dots & a_1b_0 & a_0b_0 \\ 0 & 0 & \dots & 0 & a_{n-1}b_1 & a_{n-2}b_1 & a_{n-3}b_1 & \dots & a_0b_1 & 0 \\ \vdots & & & & \vdots & \vdots & \vdots & & & \vdots \\ 0 & a_{n-1}b_{n-1} & \dots & a_2b_{n-1} & a_1b_{n-1} & a_0b_{n-1} & 0 & \dots & 0 & 0 \end{pmatrix}$$

Carry-save adder

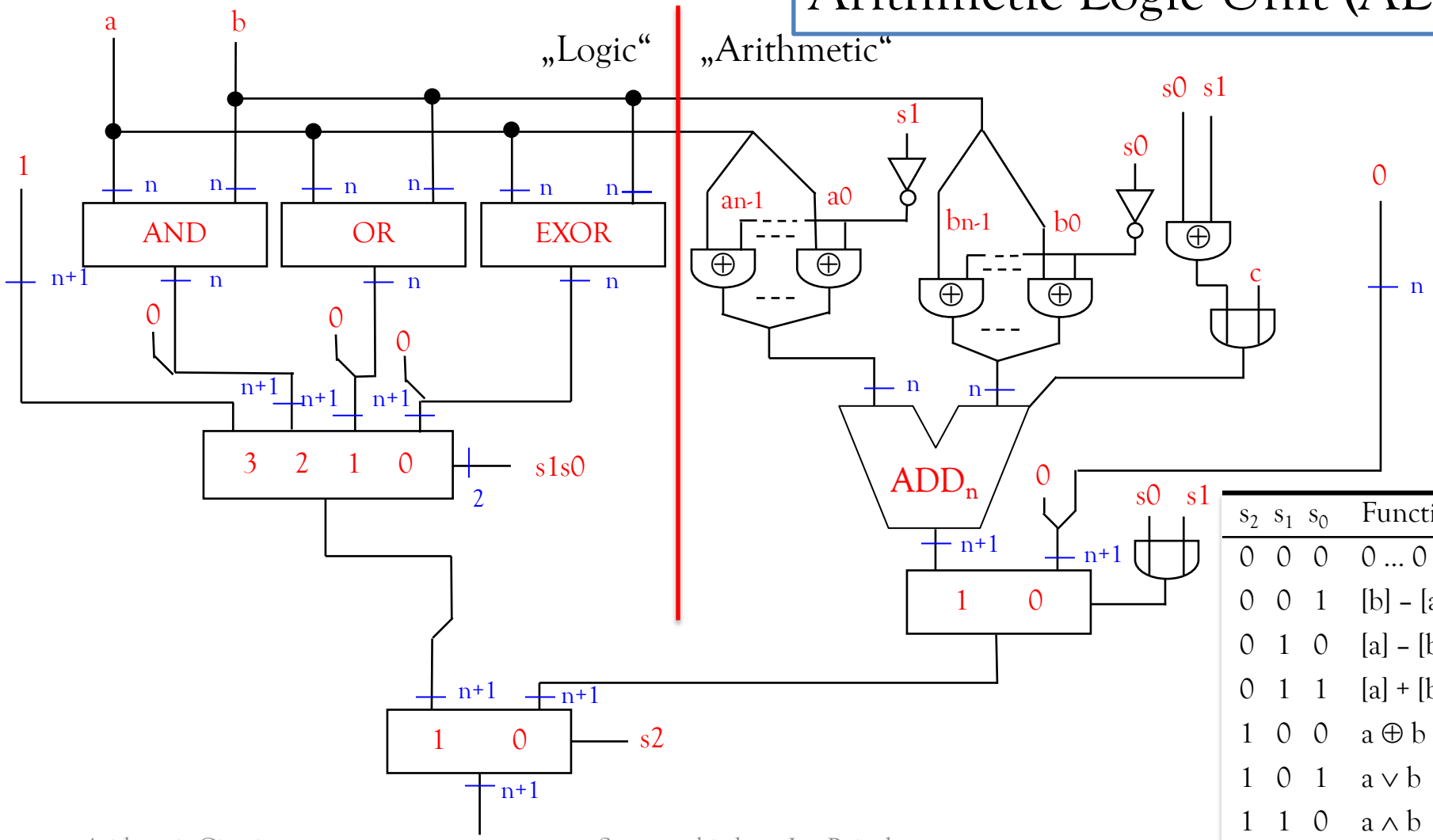


Adder stage of log-time multiplier



6. Arithmetic Circuits

Arithmetic Logic Unit (ALU)



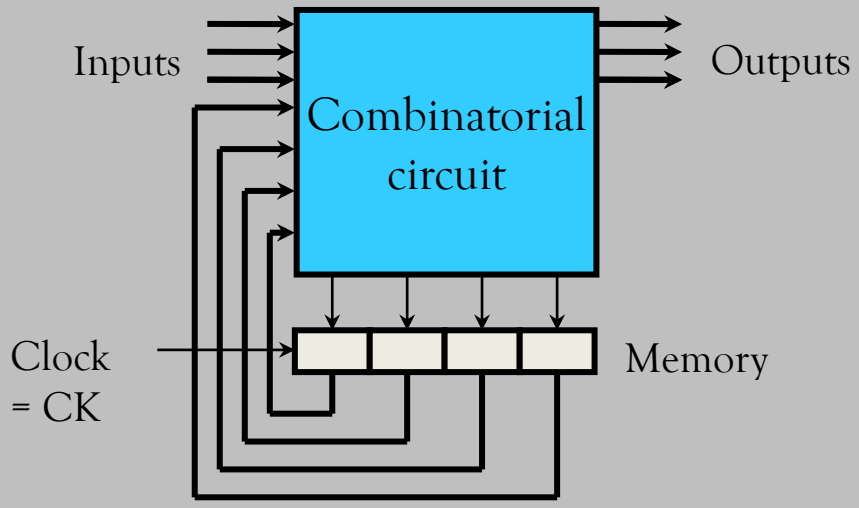
s_2	s_1	s_0	Function
0	0	0	0 ... 0
0	0	1	[b] – [a]
0	1	0	[a] – [b]
0	1	1	[a] + [b] + c
1	0	0	a ⊕ b
1	0	1	a ∨ b
1	1	0	a ∧ b
1	1	1	1 ... 1

6. Arithmetic Circuits – Key Items

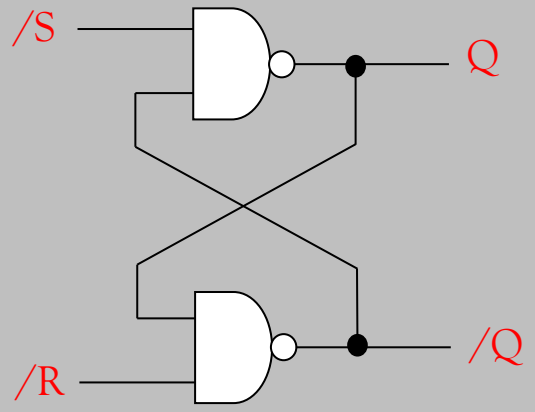
- Subtractor, combined adder/subtractor
- Multiplier
- Carry-save adder
- Arithmetic logic unit

7. Sequential Circuits

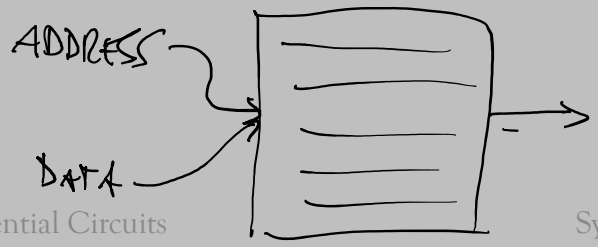
Sequential Circuit



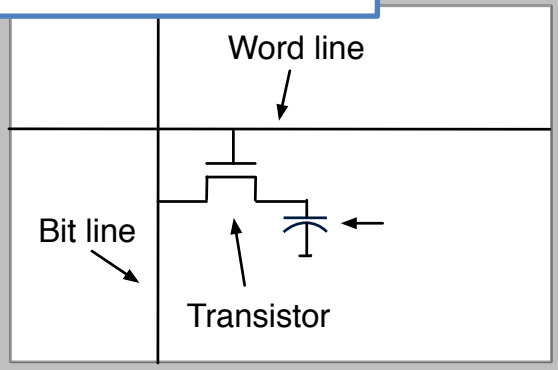
Latches and Flip-Flops



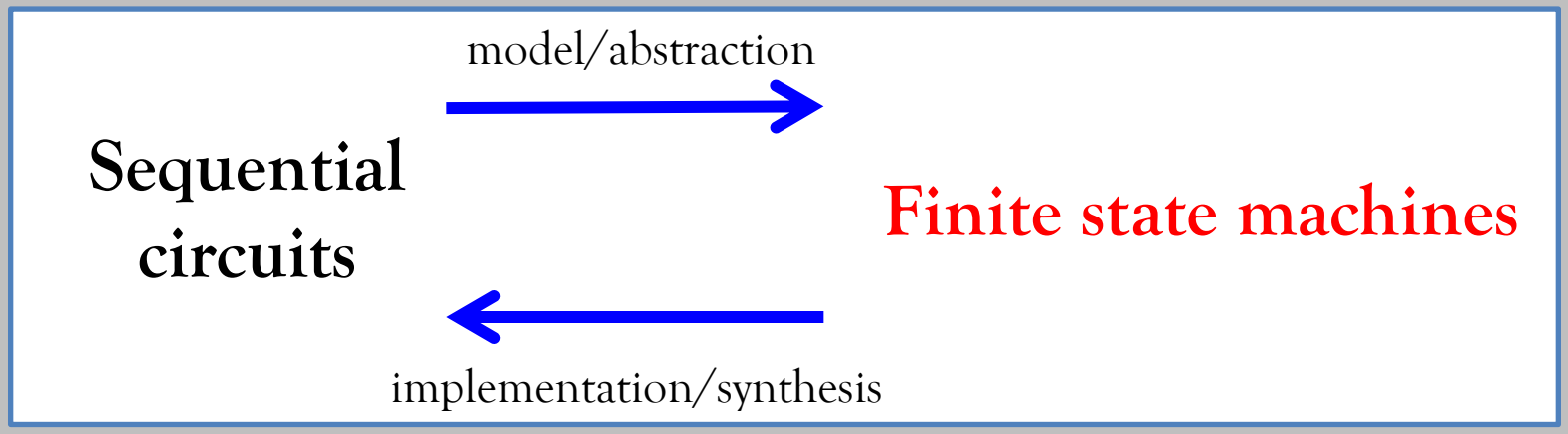
Random Access Memory



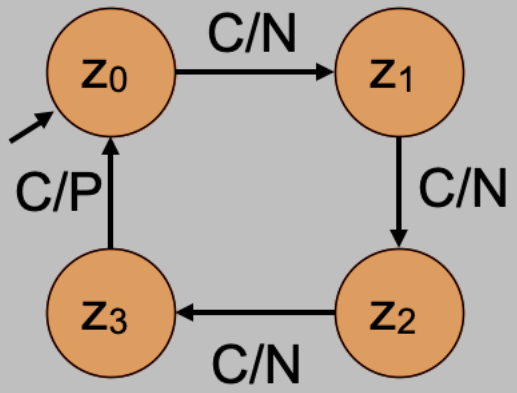
Dynamic RAM



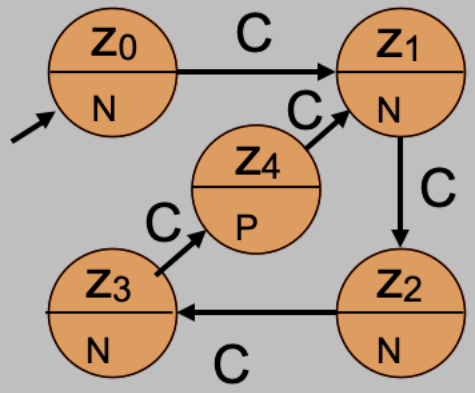
7. Sequential Circuits



Mealy Machines



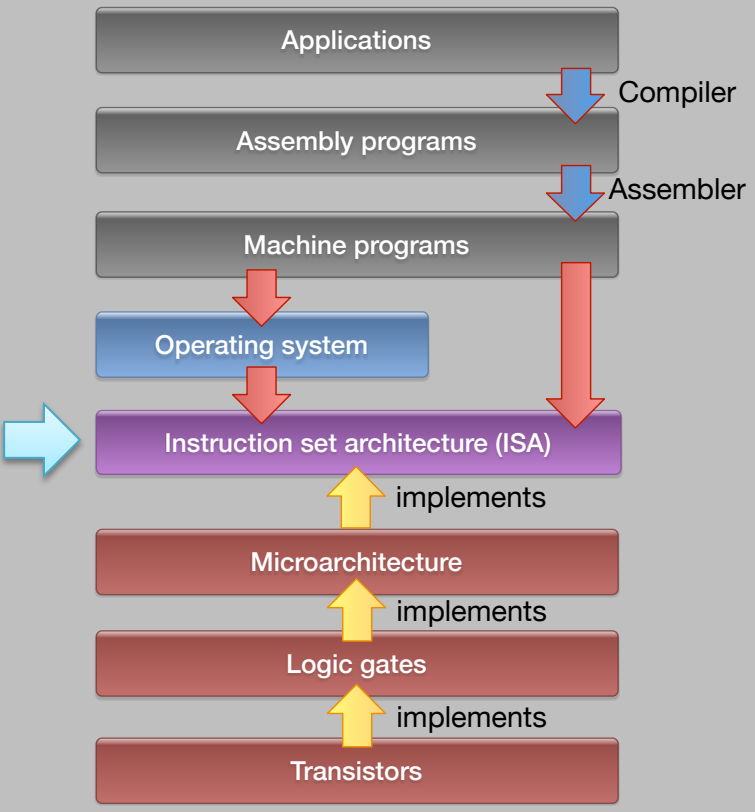
Moore Machines



7. Sequential Circuits – Key Items

- SR latch, D latch, D flip-flop
- Register
- Random access memory (RAM)
- Decoder
- Static RAM and Dynamic RAM
- Sequential circuits
- Finite state machines, Mealy and Moore machines
- Synthesis of sequential circuits

8. Instruction Set Architecture



Instruction set architecture (or just "architecture")
= set of instructions, their **encoding** and **semantics**
= „**What**" a computer computes
For example: x86, ARM

Assembly language = textual representation

Assembler +
Linker

Machine language = binary representation

MIPS instruction set

- instructions
- encoding

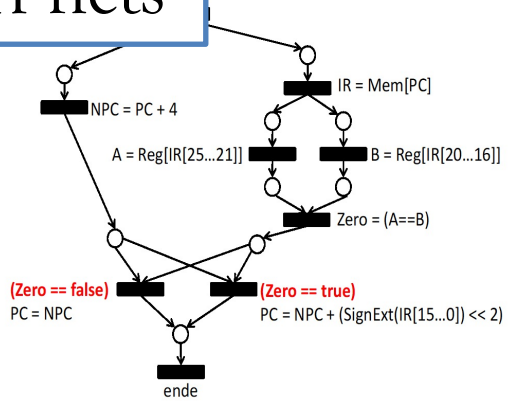
9. Microarchitecture

Microarchitecture

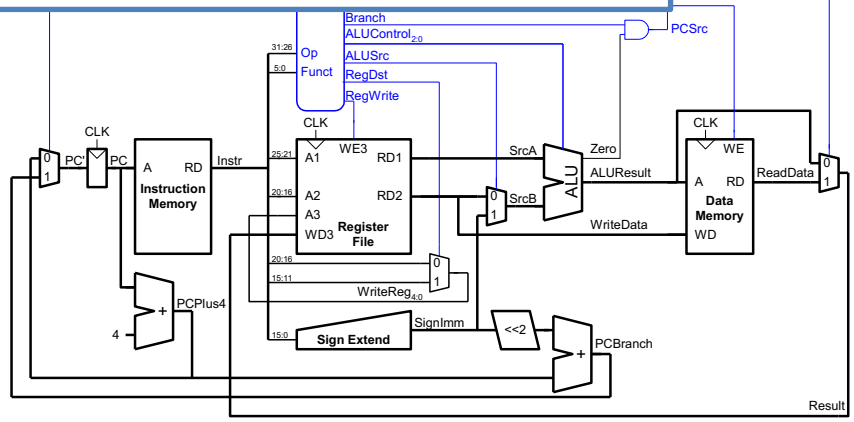
= concrete implementation of an instruction set in hardware

= „**How**“ a computer works; e.g. Intel Skylake, AMD Zen 3, Apple M1

Petri nets



Single-cycle processor



9. Microarchitecture – Key Items

- Microarchitecture
- Datapath, control
- Petri nets
- Single-cycle system
- Main decoder, ALU decoder
- ALU implementation

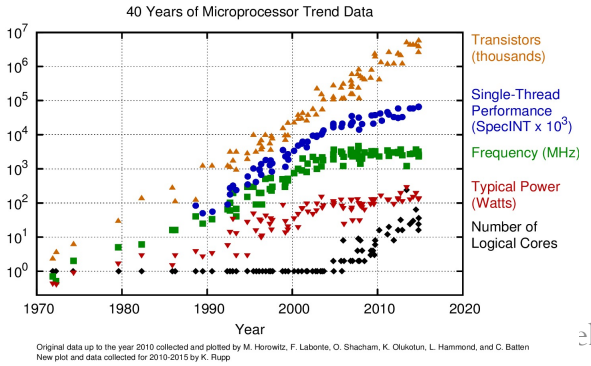
10. Performance: Basic Concepts

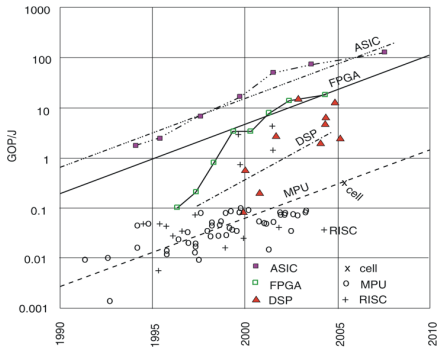
Latency = time required to perform a single task

Throughput = number of tasks performed in one time unit

Processor time = Number of executed instructions
* Cycles per instruction
* Cycle time

Technological developments

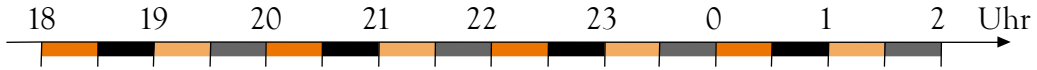


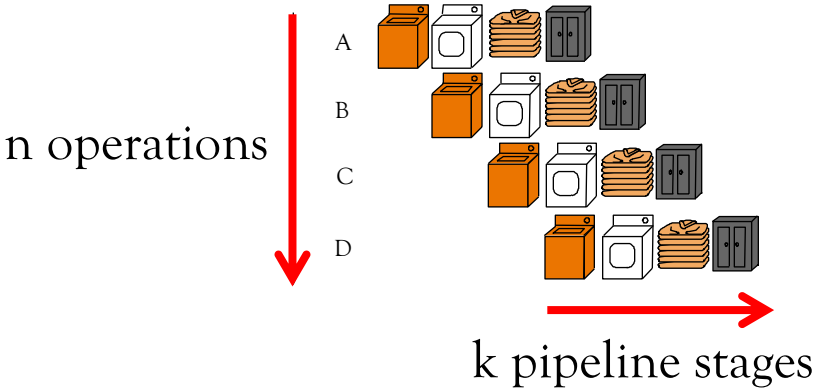


10. Performance: Basic Concepts – Key Items

- Performance definitions, latency, throughput
- Execution time, response time, processor time
- Cycles per instruction, cycle time
- Moore's law
- Reduced Instruction Set Computer (RISC), Complex Instruction Set Computer (CISC)
- Pipelining

11. Pipelining

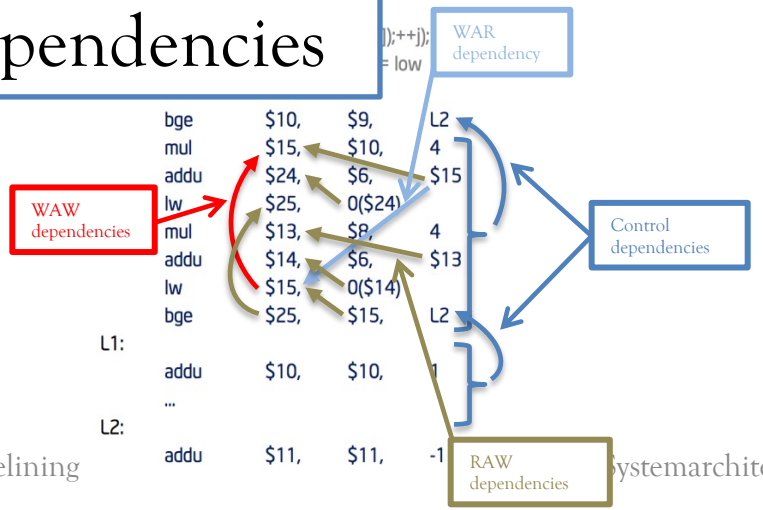




$$Speedup = \frac{n \cdot k}{k + n - 1} \xrightarrow{n \to \infty} k$$

$$\text{Efficiency} = \frac{n \cdot k}{(k + n - 1) \cdot k} = \frac{Speedup}{k} \xrightarrow{n \to \infty} 1$$

Dependencies



+ Hazards

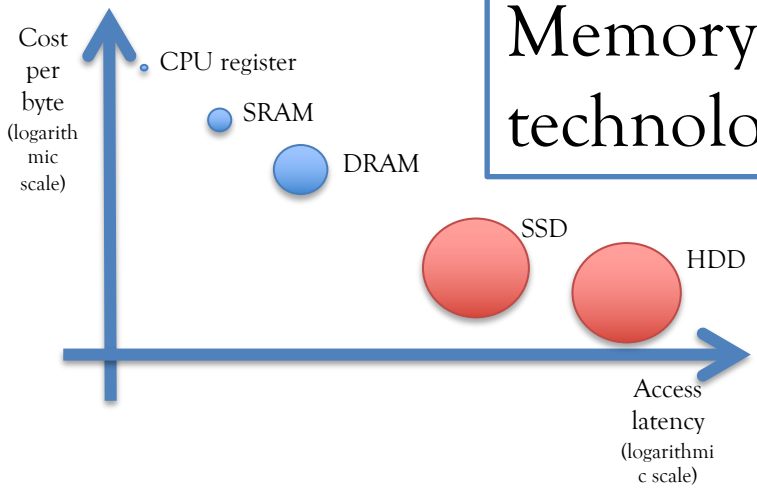
Stalls
Forwarding
Branch prediction

11. Pipelining – Key Items

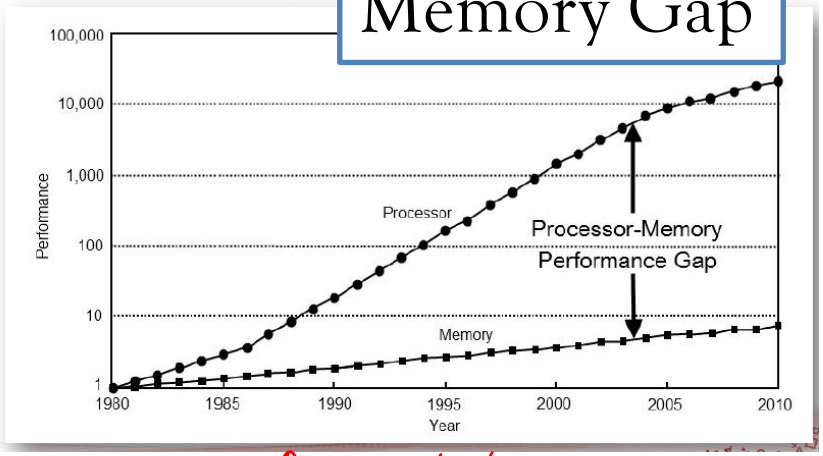
- Pipelining
- Speedup, efficiency
- Idealizing assumptions in pipelining
- Phases: IF, ID, EX, MEM, WB
- Internal and external fragmentation
- Pipelined MIPS processor
- Control dependencies
- Data dependencies, read-after write (RAW), write-after-read (WAR), write-after-write (WAW)
- Hazards
- Stalls
- Forwarding
- Branch prediction

12. Memory Hierarchy, Caches

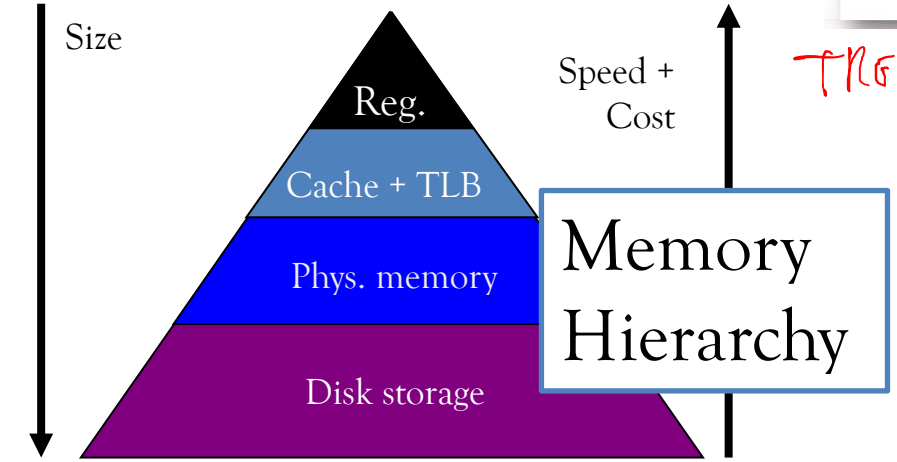
Memory technologies

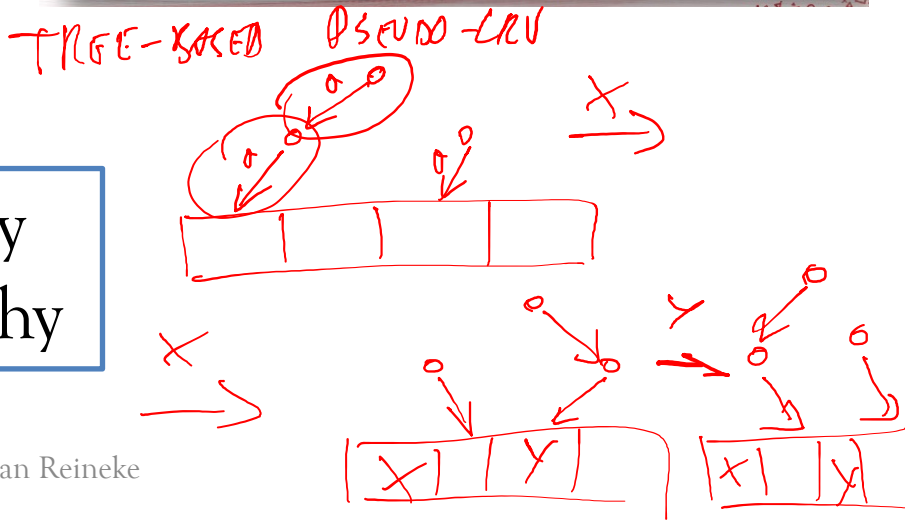


Memory Gap



Memory Hierarchy



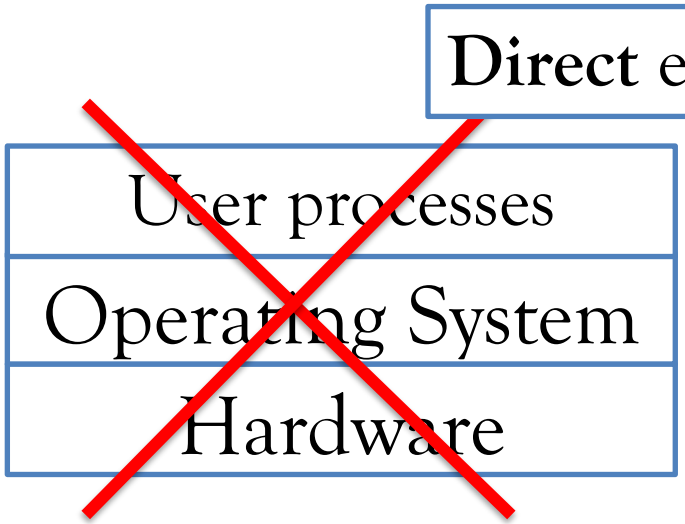


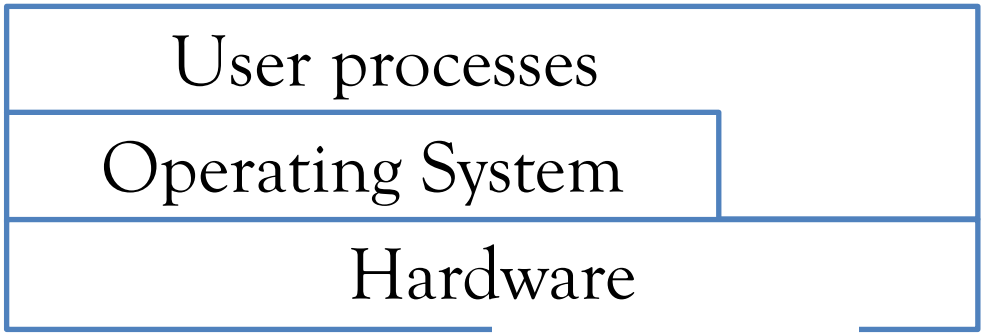
12. Memory Hierarchy, Caches – Key Items

- Memory technologies, Memory Gap
- Memory Hierarchy
- Scratchpad memory, Caches
- Fully-associative, direct-mapped, set-associative
- Replacement policy
- Optimal replacement, Farthest-in-the-Future
- Least-recently-used (LRU), First in, first out (FIFO)
- Online and offline algorithms
- Temporal and spatial locality

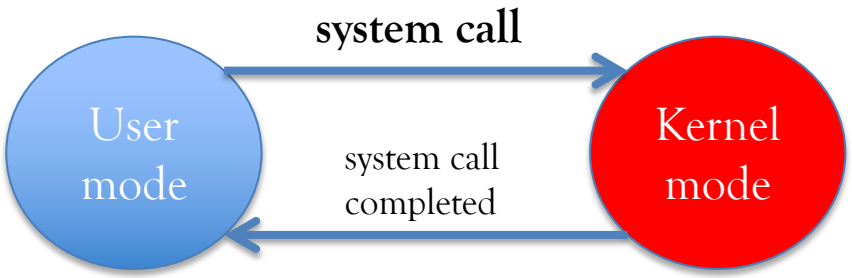
13. Virtualization: The CPU

Direct execution





Limited direct execution

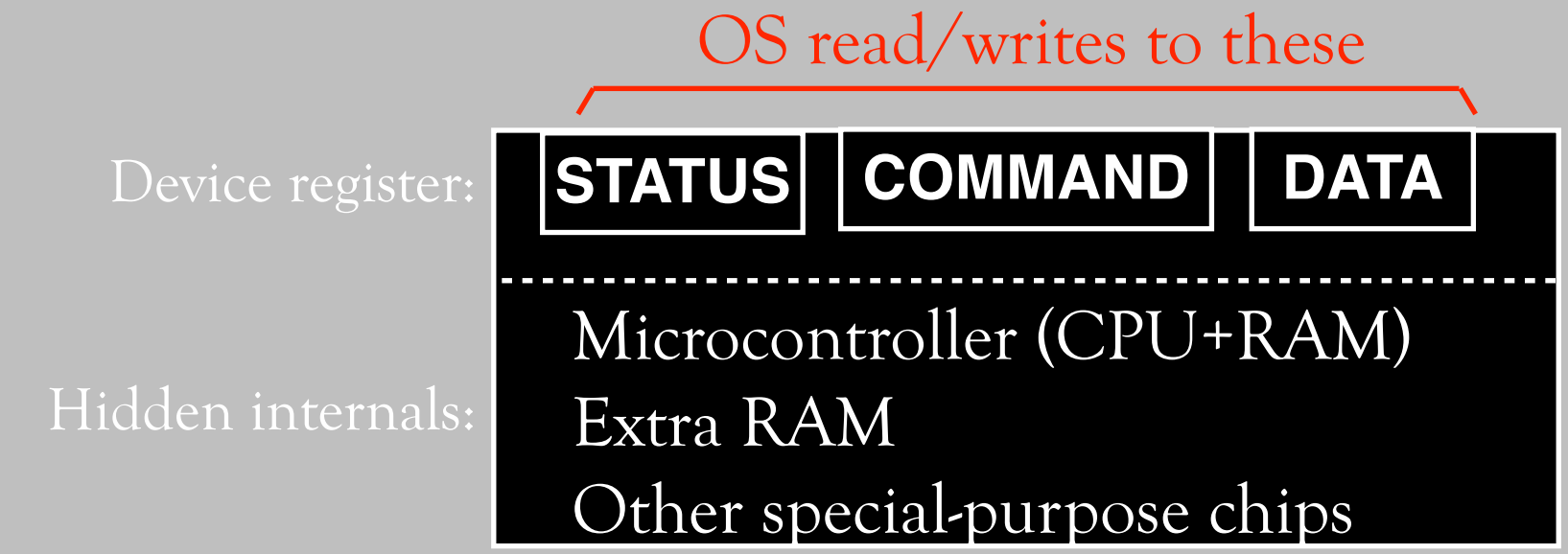


Preemptive Multitasking

13. Virtualization: The CPU – Key Items

- Process, process vs program
- Direct execution
- Restricted operations
- User mode vs kernel mode
- System calls, exception handling
- Mechanism vs policy
- Context switches
- Cooperative vs preemptive multitasking

14. Persistence: I/O Devices



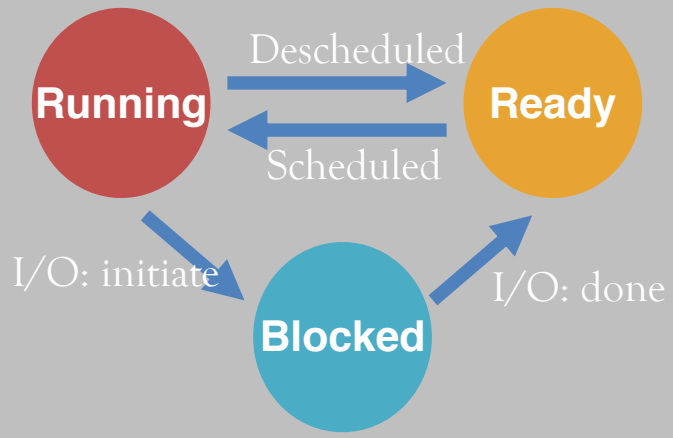
- **Status checks**: polling *vs.* interrupts
- **Data**: Programmed-IO *vs.* DMA
- **Control**: special instructions vs. memory-mapped I/O

14. Persistence: I/O Devices – Key Items

- I/O devices
- Busy waiting/polling vs interrupts
- Programmed I/O vs Direct Memory Access (DMA)
- Drivers

15. Scheduling

Process state



Performance metrics

Turnaround time
Response time
Throughput
Fairness
Meet deadlines

Scheduling policies

First Come, First Served (FCFS)
Shortest Job First (SJF)
Shortest Time-to-Completion First (STCF)
Round Robin
Multi-Level Feedback Queue (MLFQ)

Throughput
Turnaround time
Response time
Fairness
Combination

15. Scheduling – Key Items

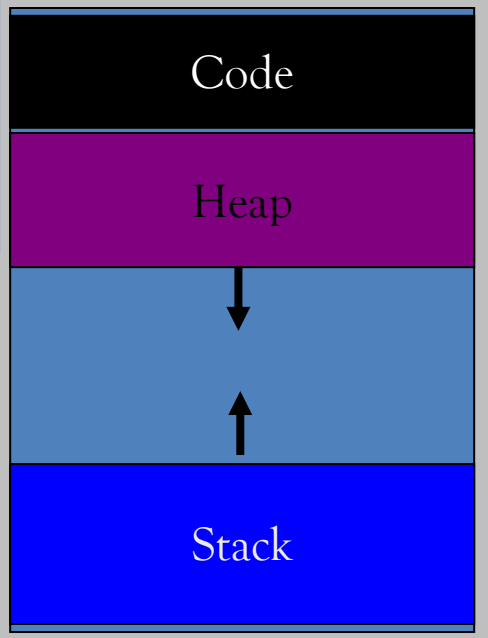
- Dispatcher vs Scheduler
- Workload, Performance metric
- Turnaround time, Response time, Throughput, Overhead, Fairness
- FIFO (also FCFS), Convoy effect
- Shortest Job First (SJF), Shortest Time-to-Completion First (STCF)
- Round Robin
- Multi-Level Feedback Queue (MLFQ)
- Starvation
- Voodoo constants

16. Memory Virtualization Foundations

Virtualization Goals

Transparency
Protection
Efficiency
Sharing

(Virtual) address space



Mechanisms for virtual memory:

1. Time sharing
2. Static relocation
3. Dynamic relocation
4. Segmentation

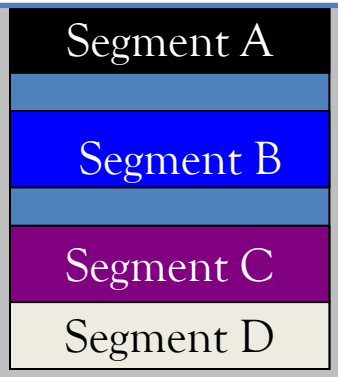
(Dis-)advantages of these?

16. Memory Virtualization Foundations – Key Items

- Transparency, protection, efficiency, sharing
- Address space
- Static: code and global data, dynamic: stack and heap
- Time sharing, Static relocation, Dynamic relocation, Segmentation
- Memory Management Unit (MMU)
- Base and bounds
- Segment table

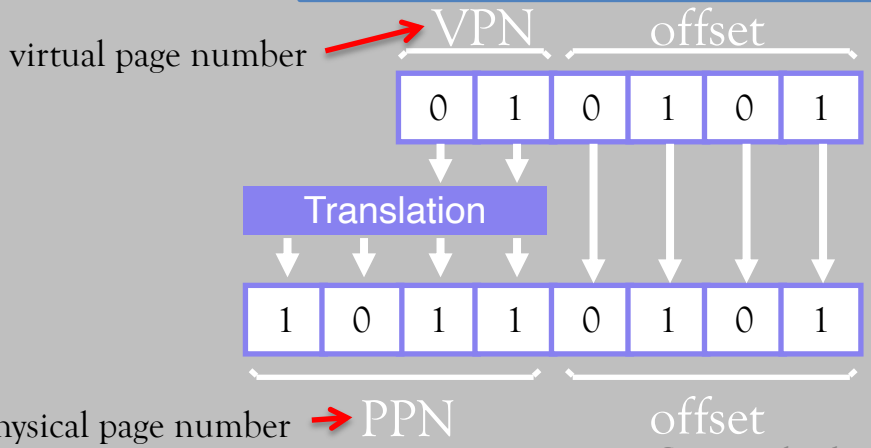
17. Paging

External and internal fragmentation

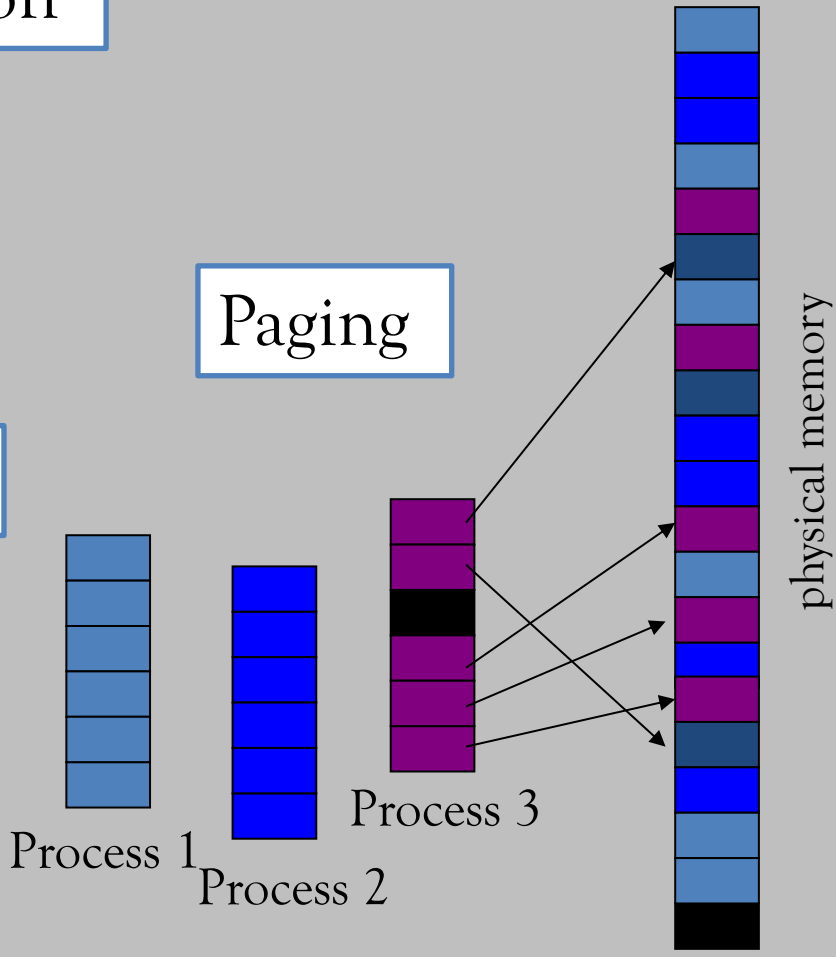


Allocated to application:

Address translation



Paging



17. Paging – Key Items

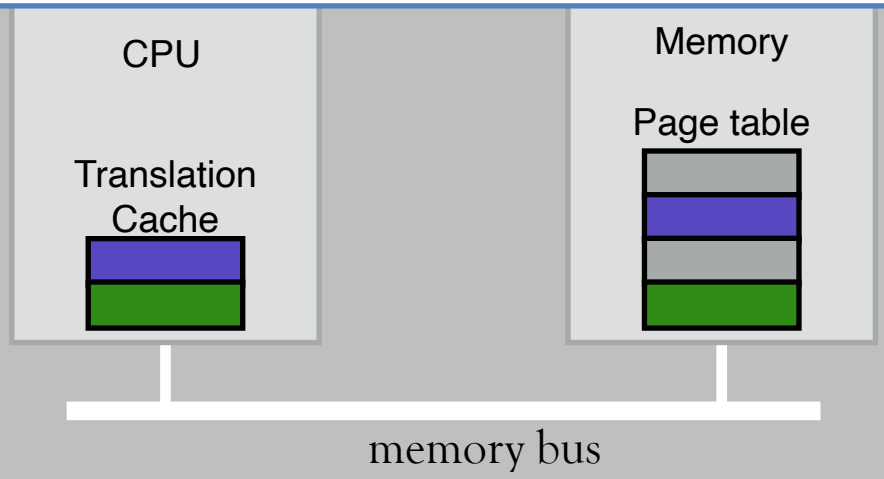
- Internal and external fragmentation
- Paging
- Pages, page frames
- Page number, frame number, page offset
- Virtual address, physical address
- Page table
- Valid bit, protection bits

18. Translation Lookaside Buffers

"Naïve" paging too slow

two physical accesses for every virtual access

Translation Lookaside Buffers



Design choices

page sizes
associativity
replacement policy

18. Translation Lookaside Buffers – Key Items

- Translation Lookaside Buffer
- Direct-mapped, fully-associative, set-associative
- Influence of page size on performance
- Influence of locality on performance
- TLB replacement policies
- Address space identifiers (ASIDs)
- TLB miss handling

19. Smaller Page Tables

Sizes of page tables

4 byte PTEs, **4 KB** pages

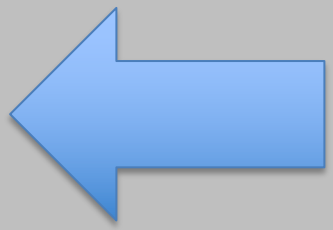
1. virt. addresses: **32 bits** — 4 MB
2. virt. addresses: **64 bits** — 2^{14} TB

VPN	valid	protection
10	1	r-x
-	0	-
23	1	rw-
-	0	-
-	0	-
-	0	-
-	0	-
…many invalid entries…		
-	0	-
-	0	-
-	0	-
-	0	-
28	1	rw-
4	1	rw-

Address spaces sparsely populated

Hierarchical page tables

Segmented page tables
Multi-level page tables

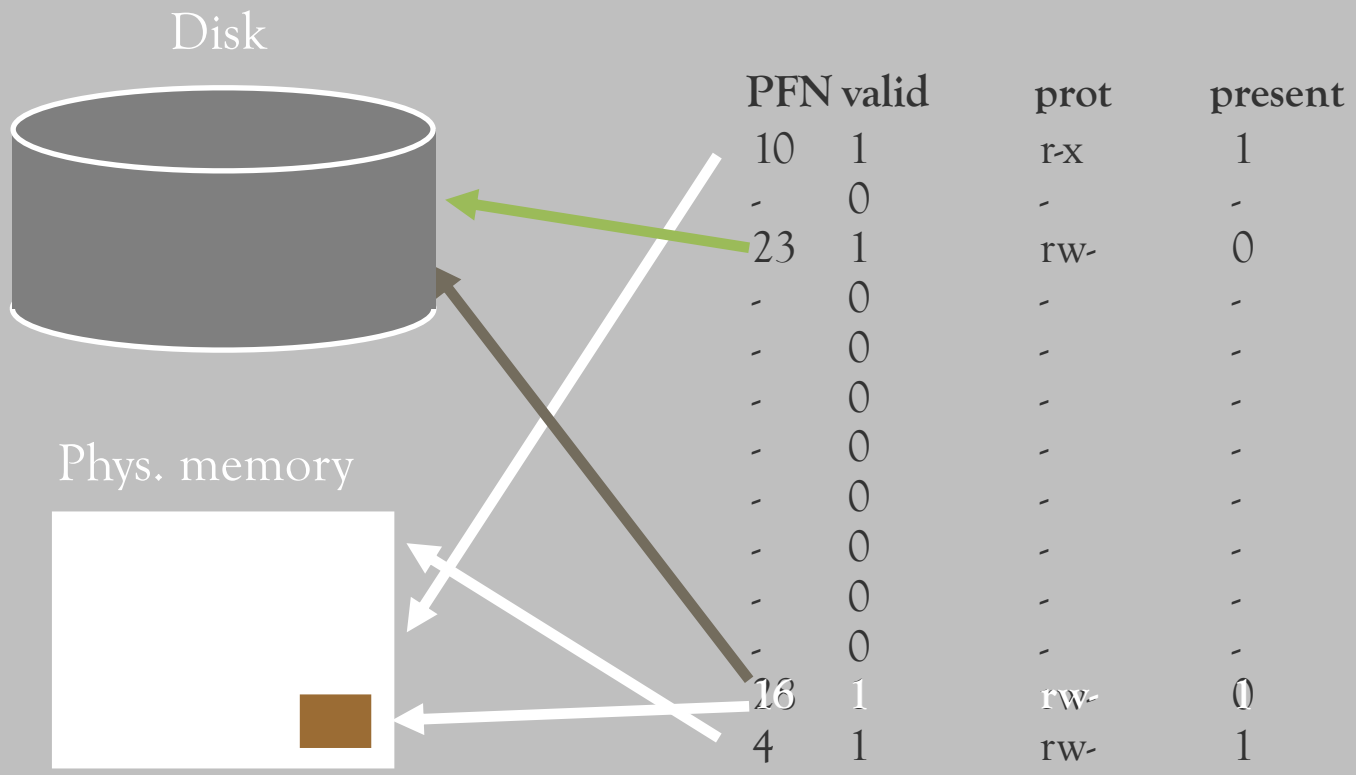


19. Smaller Page Tables – Key Items

- Invalid page table entries
- Segmented page tables
- Multi-level page tables
- Outer page, inner page
- Page tables fit within pages

20. Swapping

Pages can be in memory or on disk

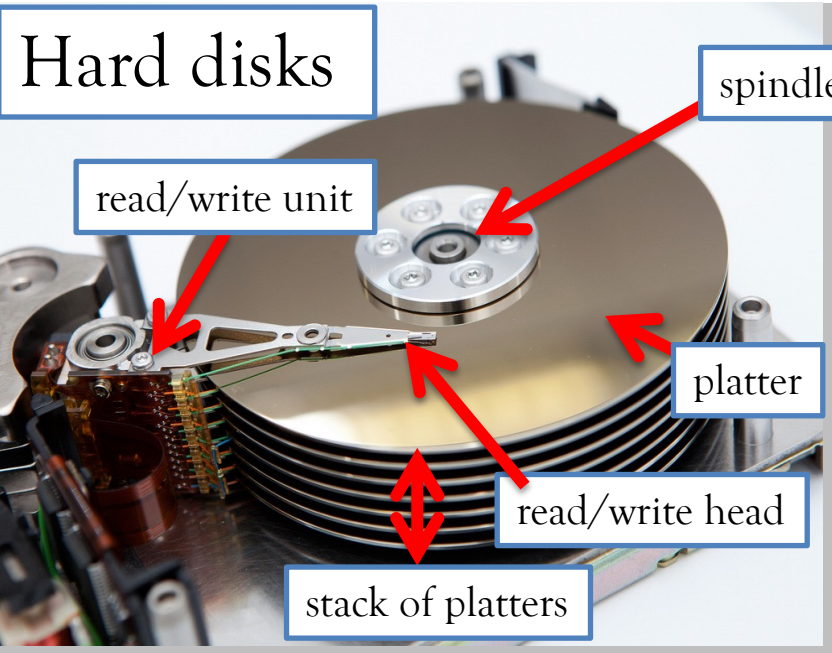


20. Swapping– Key Items

- Swapping
- Present bit in page table
- Page fault
- HW + OS cooperate on address translation
- Precise interrupts
- Page selection and Page replacement
- Demand paging, Prefetching, Hints
- Clock algorithm

21. Persistence: Disks + I/O Scheduling

Hard disks



Time to read/write

Seek → slow
Rotation → slow
Transfer time → fast

Performance depends on workload

Workload	Toshiba	Seagate Exos
Sequential	290 MB/s	261 MB/s
Random	1 MB/s	0,47 MB/s

I/O Scheduling

Shortest Positioning Time First
SCAN algorithms
Anticipatory schedulers

21. Persistence: Disks + I/O Scheduling – Key Items

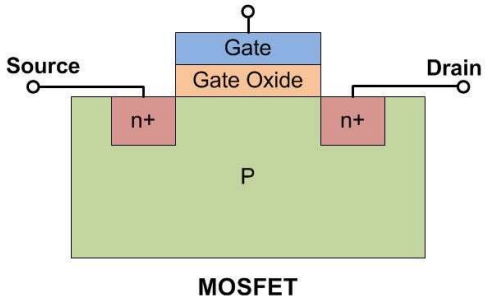
- Persistent vs volatile memory
- platter, surface, spindle, cylinder, track, sector, read/write unit, read/write head
- Seek, rotation, and transfer
- Throughput on sequential and random workloads
- Shortest Positioning Time First (SPTF), Shortest Seek Time First (SSTF)
- SCAN algorithms, Elevator algorithm, C-SCAN
- Work conservation, anticipatory schedulers

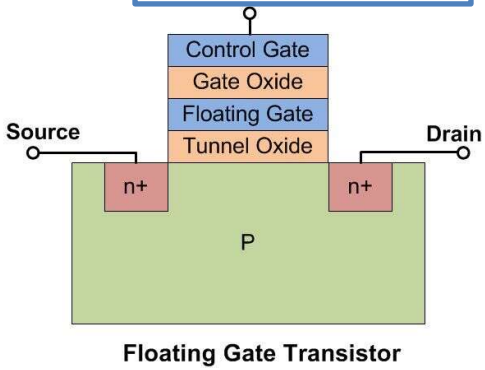
22. Persistence: Flash-based Solid State Disks

Solid-state storage devices

- No mechanical or moving parts like HDD
- Built out of transistors; but persistent unlike typical RAM

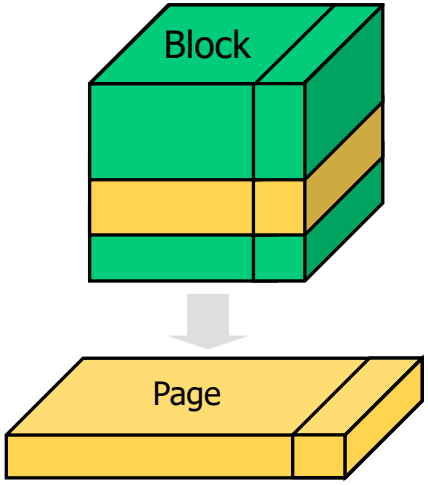
Flash cell





- electrons can be **trapped in** the floating gate
- electrons do not escape → **persistent memory**

Hierarchical organization



Read: at page granularity
Write: 1 → 0: at page granularity
Erase: 0 → 1: **only** at block granularity

22. Persistence: Flash-based Solid State Disks – Key Items

- Solid-state storage devices
- Floating-gate transistors
- Single-level cells, multi-level cells, etc.
- Basic operations: read, write, erase
- Reliability: wear out
- Out-of-place update
- Flash Translation Layer (FTL)