Language IN$FINITE$

- $INFINITE = \{w \mid L\big(\mathcal{M}(w)\big) \text{ is infinite}\}$
- We will show this is not computable by using $INFINITE$ to compute $HALT$

$\mathcal{M}(w)(x)$ **Halts?** It is infinite
$\mathcal{M}(w)(x)$ **Doesn't Halt?** It isn't infinite

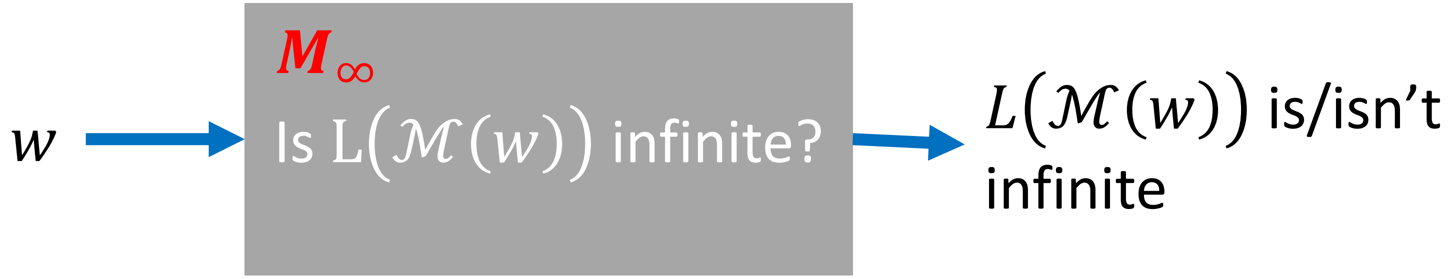
Language of M_{wx}

- If $\mathcal{M}(w)(x)$ halts:
  - M_{wx} always returns 1
  - $L(M_{wx}) = \Sigma^*$, which is infinite
- If $\mathcal{M}(w)(x)$ doesn't halt:
  - M_{wx} gets "stuck" in step 1 and never returns
  - $L(M_{wx}) = \emptyset$, which is finite

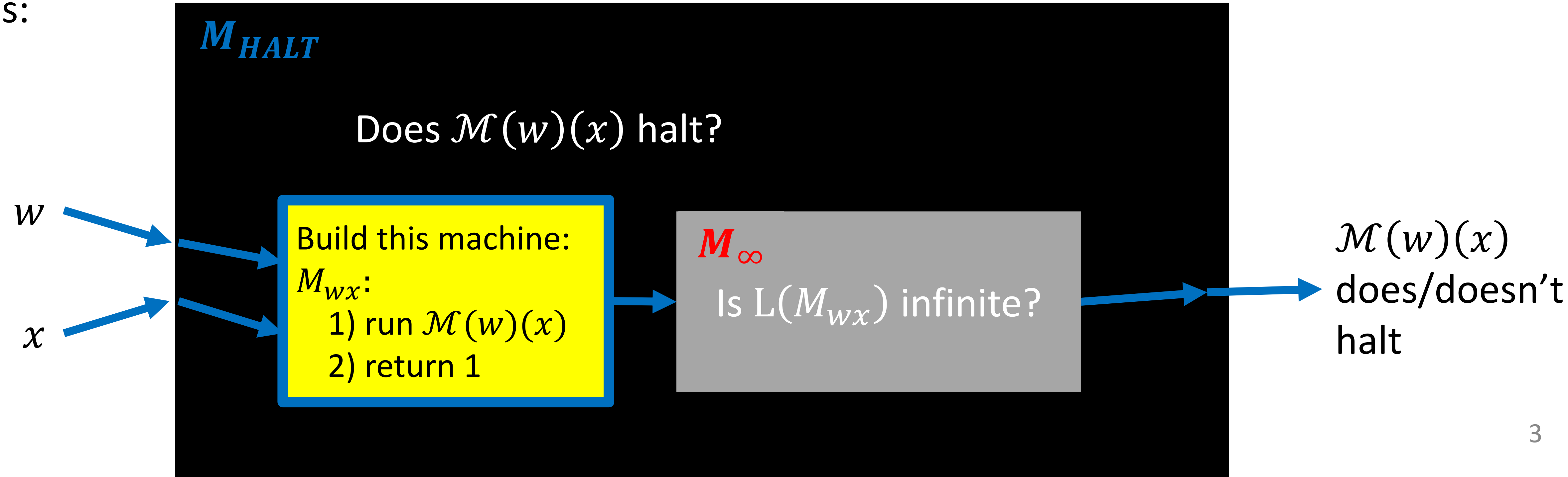
Build this machine:
M_{wx}:
1) run $\mathcal{M}(w)(x)$
2) return 1

Using $INFINITE$ to build $HALT$

Assume we have M_∞ which computes $INFINITE$:



We could then build M_{HALT} which computes $HALT$ like this:



Reduction

Problem **know** is impossible

A

$HALT$

Does $\mathcal{M}(w)$ halt on input x?

Build (but don't run) M_{wx}

Problem we **think** is impossible

B

$INFINITE$

Is $\mathrm{L}(\mathcal{M}(w))$ infinite?

Assume

Give M_{wx} to the $INFINITE$ solver, then answer the same

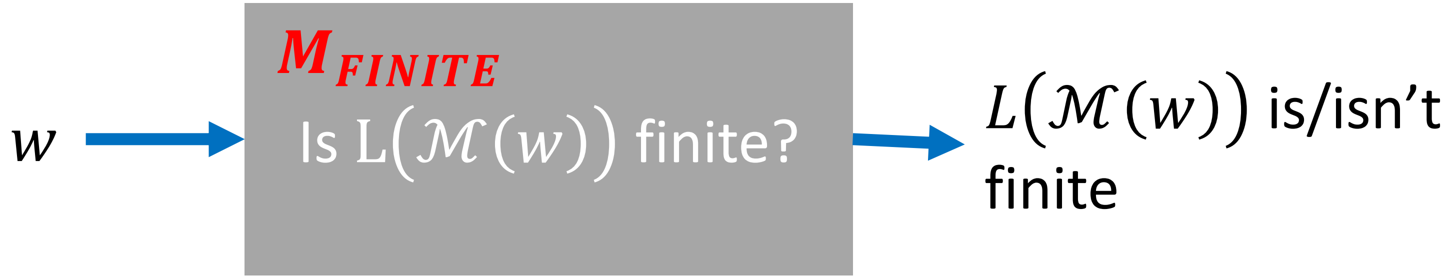
M_{HALT} computes $HALT$

$INFINITE$ solver exists

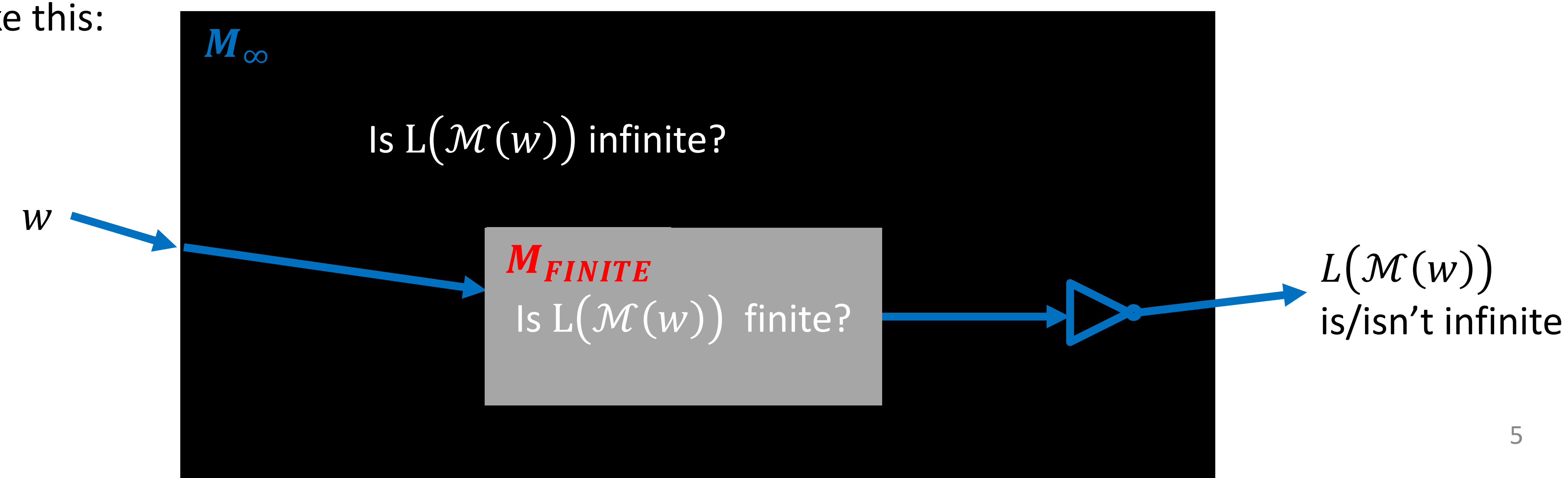
Reduction

Using $FINITE$ to build $INFINITE$

Assume we have M_{FINITE} which computes $FINITE$:



We could then build M_∞ which computes $INFINITE$ like this:



Language $NonReg$

- $NonReg = \{w \mid L(\mathcal{M}(w)) \text{ is non} - \text{regular}\}$
- We will show this is not computable by using $NonReg$ to compute $HALT$

$\mathcal{M}(w)(x)$ **Halts?** It is non-regular
$\mathcal{M}(w)(x)$ **Doesn't Halt?** It isn't non-regular

Language of M_{wx}

Build this machine:
$M_{wx}(y)$:
1) run $\mathcal{M}(w)(x)$
2) return $\mathrm{MAJ}(y)$

- If $\mathcal{M}(w)(x)$ halts:
  - M_{wx} returns 1 if $\mathrm{MAJ}(y) = 1$
  - $L(M_{wx}) = MAJ(y)$, which is non-regular
- If $\mathcal{M}(w)(x)$ doesn't halt:
  - M_{wx} gets "stuck" in step 1 and never returns 1
  - $L(M_{wx}) = \emptyset$, which isn't non-regular

Using $NonReg$ to build $HALT$

Assume we have M_{NR} which computes $NonReg$:

w

M_{NR}

Is $L(\mathcal{M}(w))$ non-regular?

$L(\mathcal{M}(w))$ is/isn't non-regular

We could then build M_{HALT} which computes $HALT$ like this:

M_{HALT}

Does $\mathcal{M}(w)(x)$ halt?

w

x

Build this machine:

$M_{wx}(y)$:

1) run $\mathcal{M}(w)(x)$
2) return MAJ(y)

M_{NR}

Is $L(\mathcal{M}(w))$ non-regular?

$\mathcal{M}(w)(x)$ does/doesn't halt

Reduction

A

$HALT$

Does $\mathcal{M}(w)$ halt on input x?

Describe that machine M_{wx}

B

$NonReg$

Is $L(\mathcal{M}(w))$ non-regular?

Assume

$NonReg$ solver exists

Give M_{wx} to the $NonReg$ solver, then answer the same

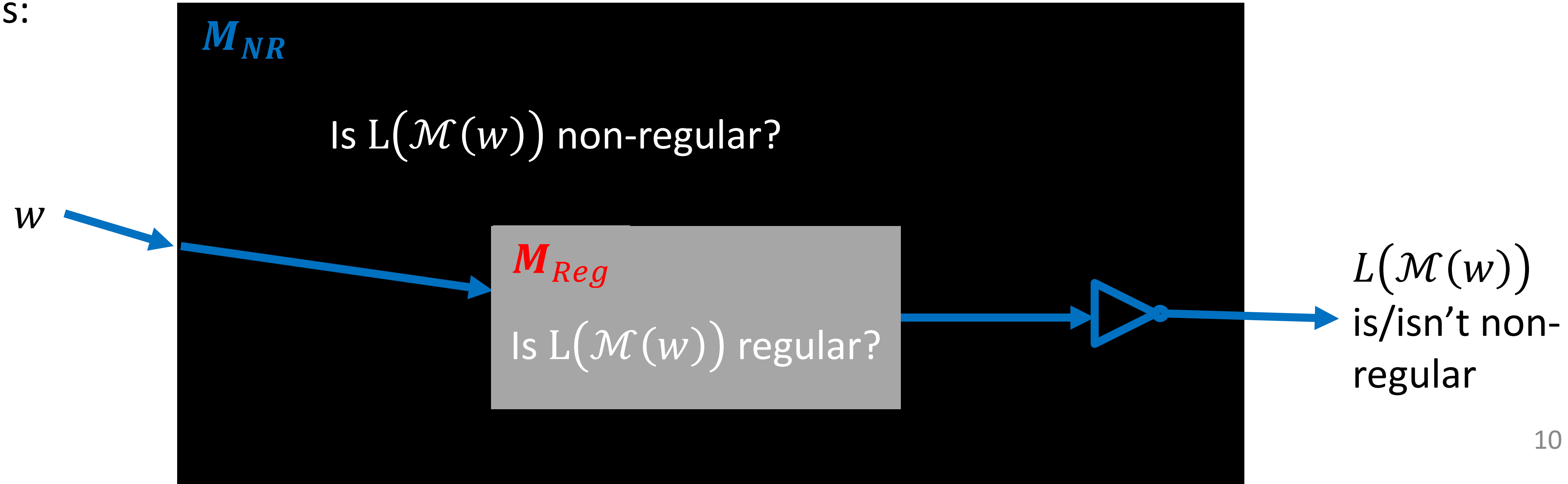
M_{HALT} computes $HALT$

Reduction

Using Reg to build $NonReg$

Assume we have M_{Reg} which computes $NonReg$:

We could then build M_{NR} which computes $NonReg$ like this:



Language $Accept101$

- $Accept101 = \{w \mid \mathcal{M}(w)(101) = 1\}$
- We will show this is not computable by using$Accept101$ to compute $HALT$

$\mathcal{M}(w)(x)$ **Halts?** It **does** accept 101
$\mathcal{M}(w)(x)$ **Doesn't Halt?** It **doesn't** accept101

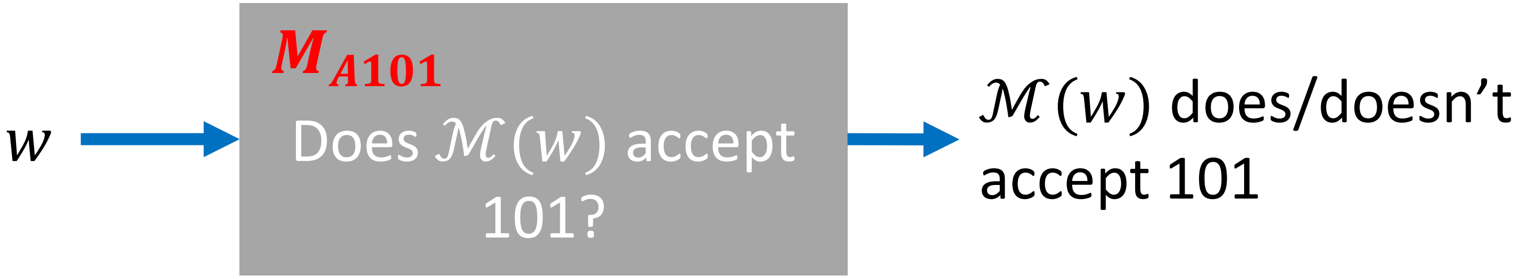
Building M_{wx}

Build this machine:
$M_{wx}(y)$:
1) run $\mathcal{M}(w)(x)$
2) return $y == 101$

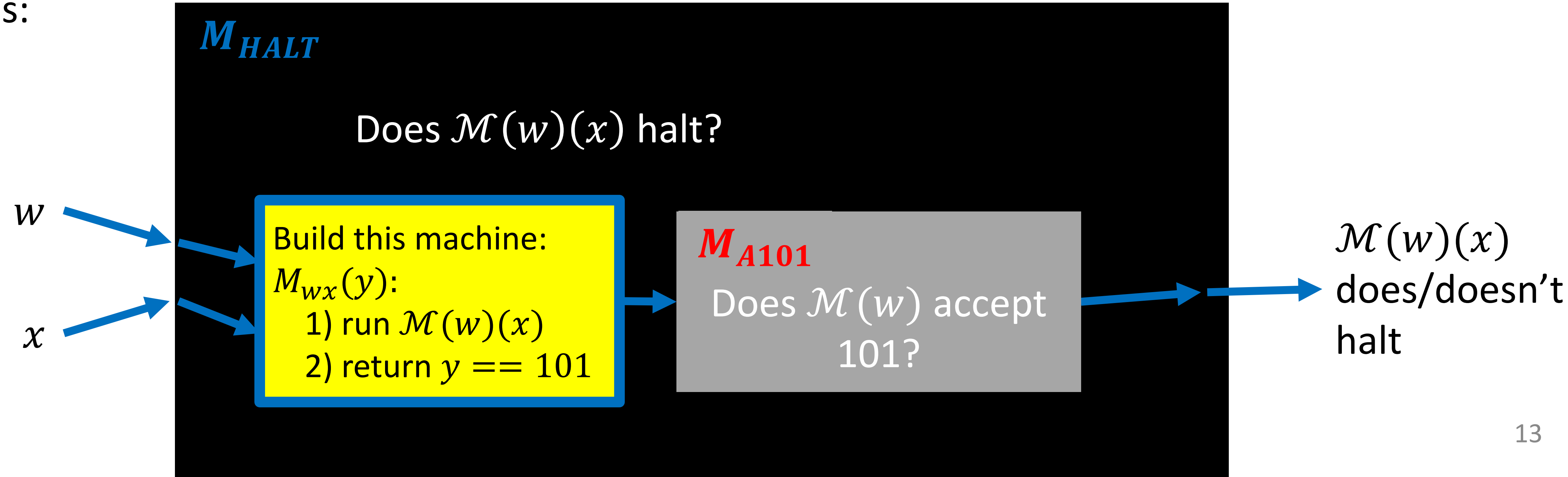
- If $\mathcal{M}(w)(x)$ halts:
  - M_{wx} returns 1 if $y == 101$
  - $L(M_{wx}) = \{101\}$, so it does accept 101
- If $\mathcal{M}(w)(x)$ doesn't halt:
  - M_{wx} gets "stuck" in step 1 and never returns 1
  - $L(M_{wx}) = \emptyset$, so it doesn't accept 101

Using $Accept101$ to build $HALT$

Assume we have M_{A101} which computes $Accept101$:



We could then build M_{HALT} which computes $HALT$ like this:



$Accept101$ Reduction

A

$HALT$

Does $\mathcal{M}(w)$ halt on input x?

Describe that machine M_{wx}

B

$Accept101$

Is $101 \in \mathrm{L}(\mathcal{M}(w))$?

Assume

Give M_{wx} to the $Accept101$ solver, then answer the same

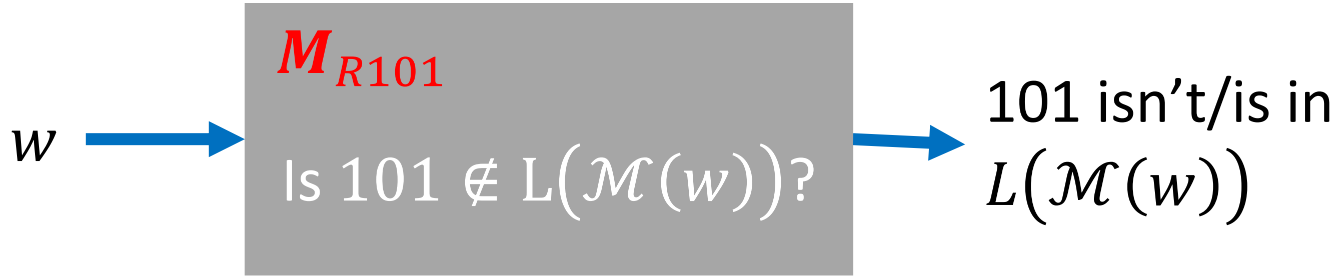
M_{HALT} computes $HALT$

$Accept101$ solver exists

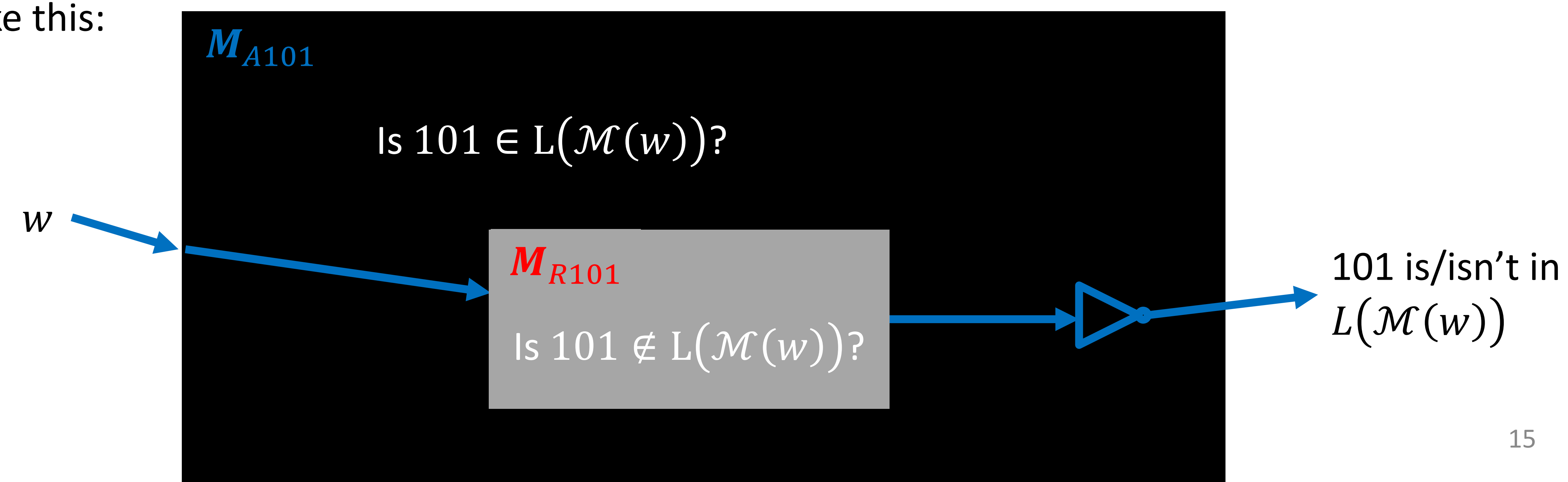
Reduction

Using $Reject101$ to build $Accept101$

Assume we have M_{R101}
which computes $Reject101$:



We could then build M_{A101}
which computes $Accept101$
like this:

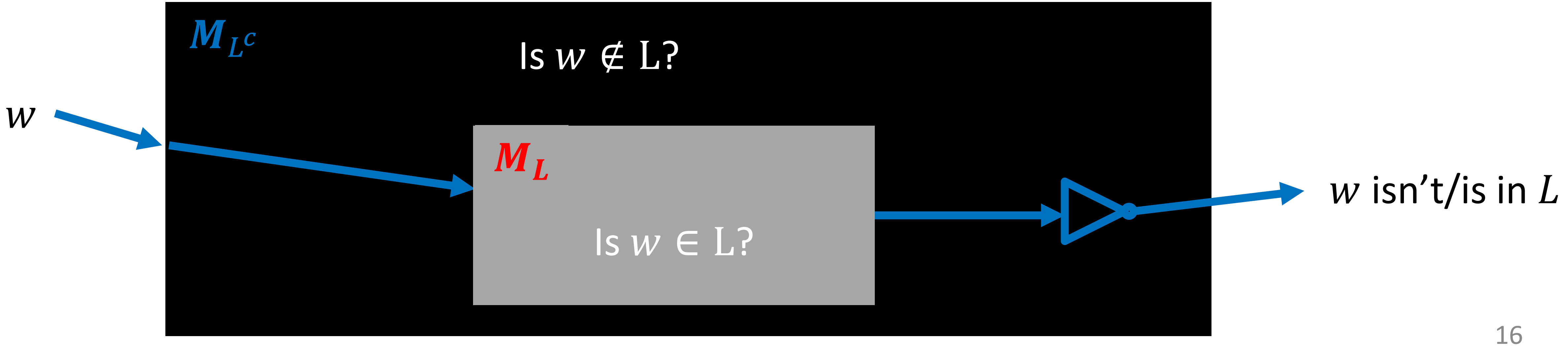


Computability and non-computability closed under complement

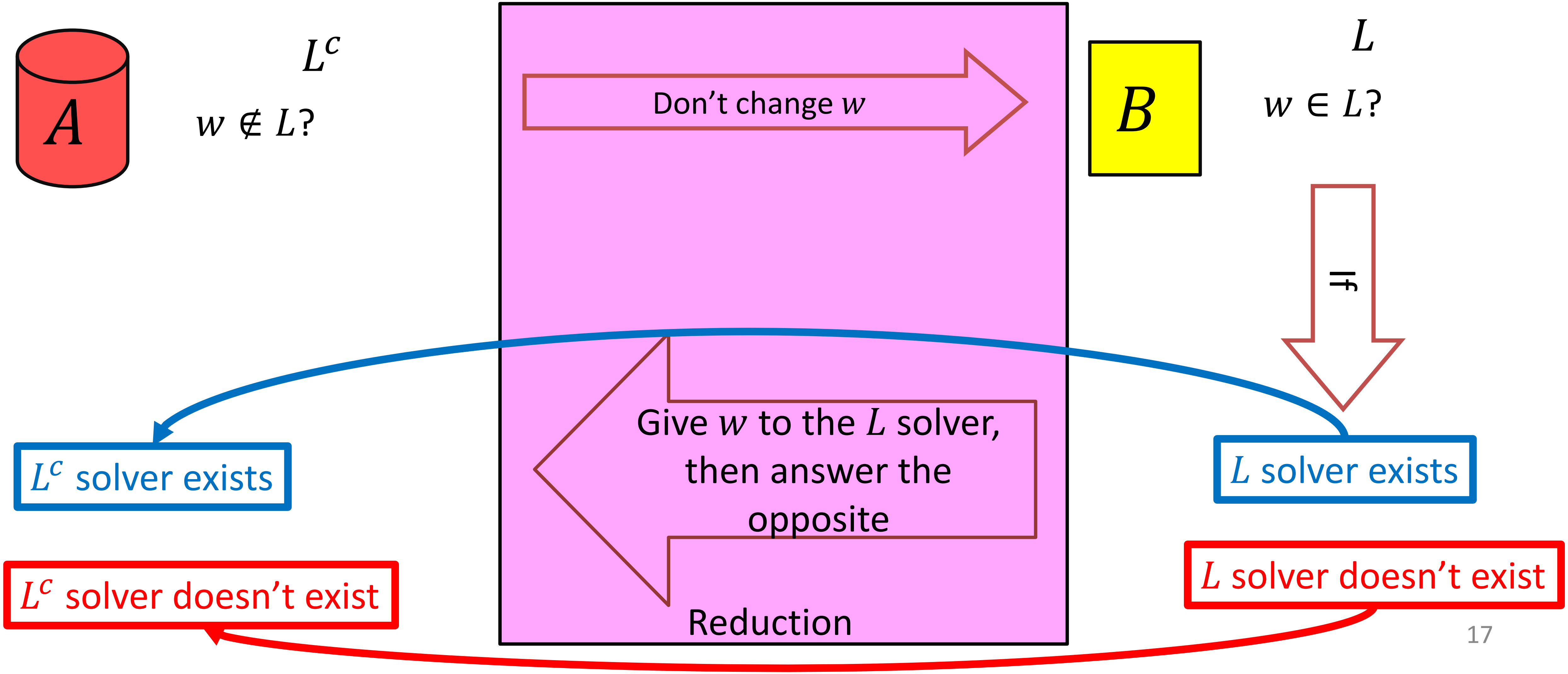
- If L is computable, then L^c is computable

Assume we have M_L which computes L:

We could then build M_{L^c} which computes L^c like this:



Complement Reduction



Sematic Property

- Turing machines M, M' are **Functionally Equivalent** if $\forall x \in \Sigma^*, M(x) == M(x')$
  - i.e. they compute the same function/language
- A **Semantic Property** of a Turing machine is one that depends only on the input/output behavior of the machine
  - Formally, if P is semantic, then for machine M, M' that are functionally equivalent, $P(M) == P(M')$
  - If M, M' have the same input/output behavior, and P is a semantic property, then either bother M and M' have property P, or neither of them do.

Examples

- These properties are Semantic:
  - Is the language of this machine finite?
  - Is the language of this machine Regular?
  - Does this machine reject 101?
  - Does this machine return 1001 for input 001?
  - Does this machine only ever return odd numbers?
  - Is the language of this machine computable?
- These properties are not Semantic:
  - Does this machine ever overwrite cell 204 of its tape?
  - Does this machine use more than 3102 cells of its tape on input 101?
  - Does this machine take at least 2020 transitions for input ε?
  - Does this machine ever overwrite the ∇ symbol?

Rice's Theorem

- For any Semantic property P of Turing Machines, either:
  - Every Turing machine has property P
  - No Turing machines have property P
  - P is uncomputable

P is "trivial"

- In other words:
  - If P is semantic, and computable, then one of these two machines computes it:

Return 1

Return 0

Proof of Rice's Theorem

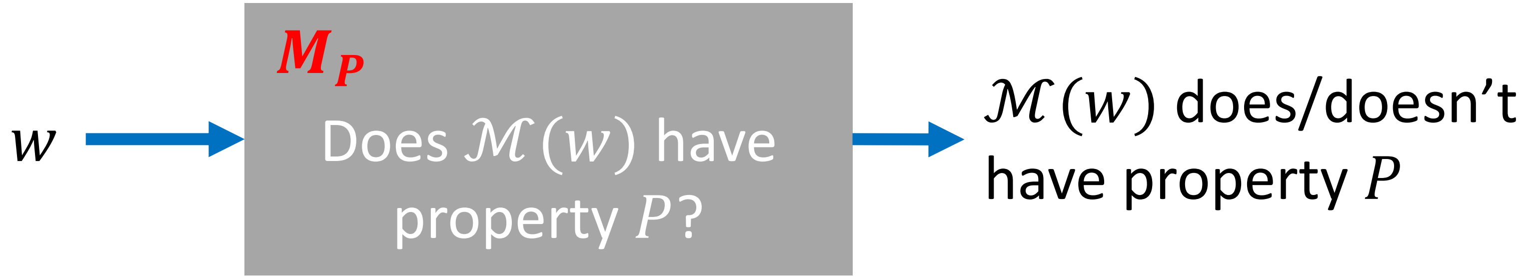
- Let P be a semantic property of a Turing machine
- Assume $M_\emptyset$ (a machine whose language is ∅) doesn't have property P (otherwise substitute $\neg P$, then answer opposite)
- Let M_P be a machine that does have property P
- Idea:
  - If $\mathcal{M}(w)(x)$ halts, $L(M_{wx}) = L(M_{has\ P})$
  - If $\mathcal{M}(w)(x)$ doesn't halt, $L(M_{wx}) = \mathrm{L}(M_\emptyset) = \emptyset$
  - $L(M_{wx})$ has property P if and only if $\mathcal{M}(w)(x)$ halts

$\mathcal{M}(w)(x)$ **Halts?** It **has** property P
$\mathcal{M}(w)(x)$ **Doesn't Halt?** It **doesn't have** property P

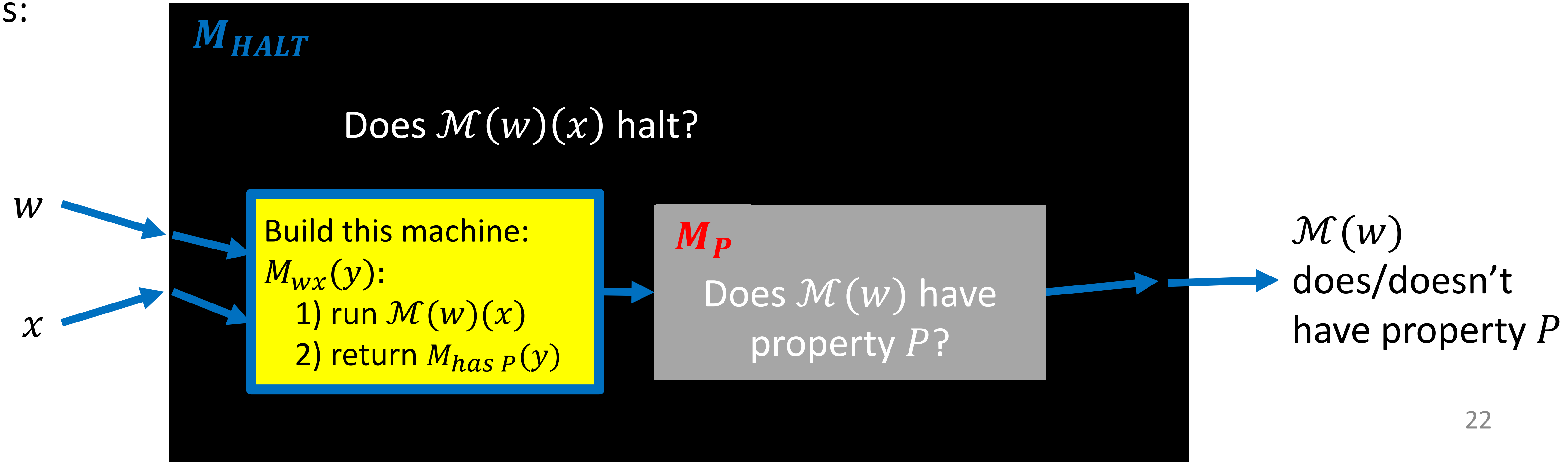
Build this machine:
$M_{wx}(y)$:
1) run $\mathcal{M}(w)(x)$
2) return $M_{has\ P}(y)$

Using P to build $HALT$

Assume we have M_P which computes P:



We could then build M_{HALT} which computes $HALT$ like this:



What if P is "trivial"?

- If P applies to all Turing machines:
  - It applies to $M_{\emptyset}$, so we'll consider $\neg P$ which applies to no Turing machines
- If P applies to no Turing machines:
  - $M_{Has\ P}$ can't exist

Build this machine:
$M_{wx}(y)$:
1) run $\mathcal{M}(w)(x)$
2) return $M_{Has\ P}(y)$

Using Rice’s Theorem

- These properties are Semantic:
  - Is the language of this machine finite?
  - Is the language of this machine Regular?
  - Does this machine reject 101?
  - Does this machine return 1001 for input 001?
  - Does this machine only ever return odd numbers?
  - Is the language of this machine computable?
- These properties are not Semantic:
  - Does this machine ever overwrite cell 204 of its tape?
  - Does this machine use more than 3102 cells of its tape on input 101?
  - Does this machine take at least 2020 transitions for input ε?
  - Does this machine ever overwrite the ∇ symbol on input ε?

$Steps2020$

- $Steps2020 = \{w | \mathcal{M}(w) \text{ takes at least 2020 steps}\}$
- Is $Steps2020$ computable?

$Overwrite\nabla$

- $Overwrite\nabla = \{\mathrm{w} \mid \mathcal{M}(w) \text{ overwrites } \nabla \text{ on input } \varepsilon\}$
- Is $Overwrite\nabla$ computable?

CS3102 Theory of Computation

www.cs.virginia.edu/~njb2b/cstheory/s2020

Warm up:

To measure the “cost” of computing something, what would units should we use?

Units

Computing Cost

- What do we actually care about with computing cost?

Notions of function "difficulty"

- Can an algorithm for function f be implemented using this computing model?
  - NAND-CIRC: answer is YES iff f is finite
  - FSA: answer is YES if function doesn't require "memory"
  - TM: Answer is NO for $HALT_{TM}, FINITE,$ ... (and many other things)
- How efficient is an algorithm for function f implemented using this computing model?
  - NAND-CIRC: How many gates?
  - FSA: (we never talked about this)
  - TM: How many transitions are required (time)? How many tape cells are required (space)?

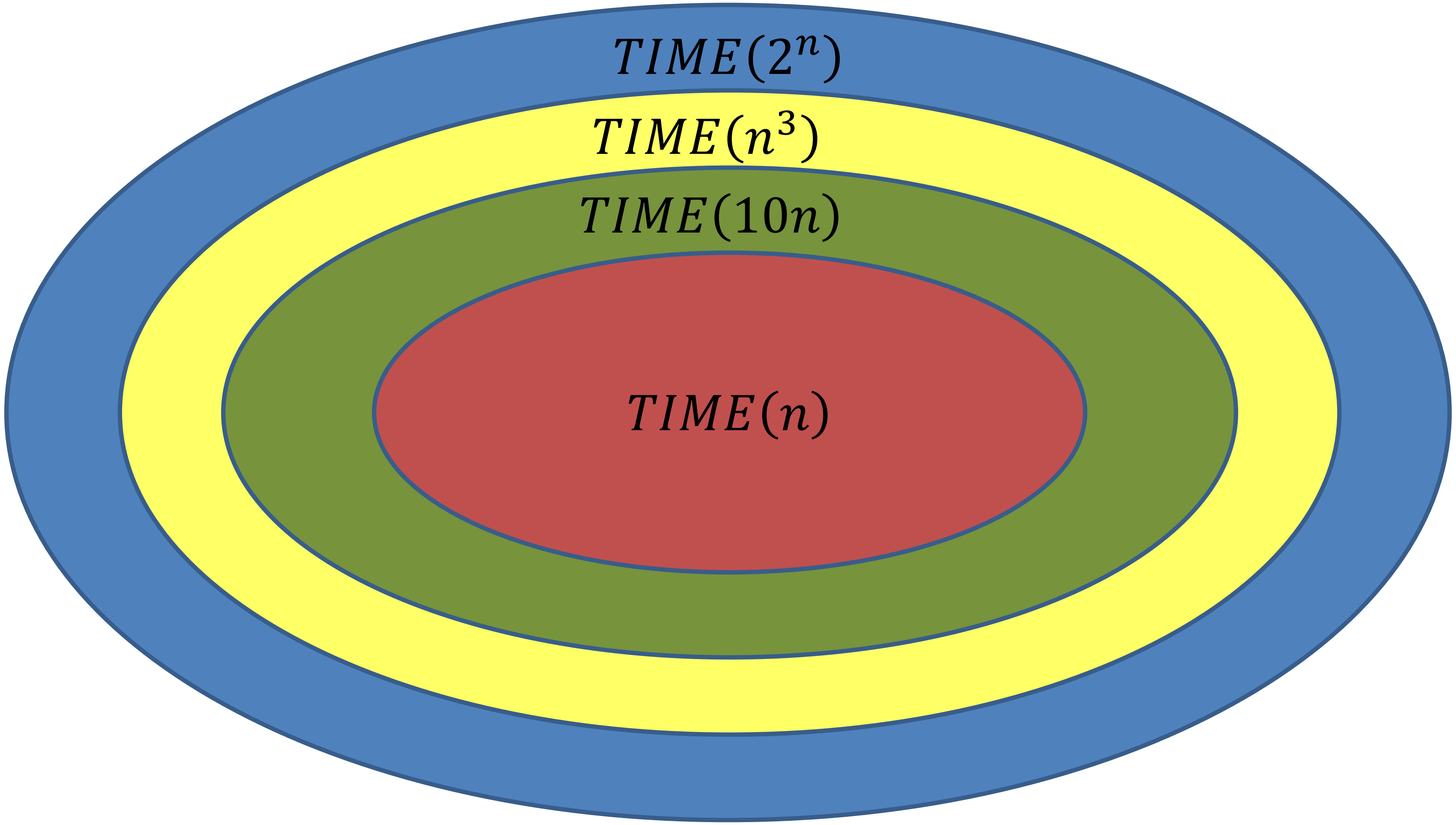
Larger inputs = More time

- Run time is not measured by a number, but a function.
- **Running time:** $T(n)$ is a function mapping naturals to naturals. We say $F: \{0,1\}^* \rightarrow \{0,1\}^*$ is computable in $T(n)$ time if there exists a TM M s.t. for every large n and ever input $x \in \{0,1\}^n$, M halts after at most $T(n)$ steps and outputs $F(x)$.
- $TIME\big(T(n)\big)$ represents the set of boolean functions computable within $T(n)$ time

Examples

- How long will XOR take on a Turing Machine?
  - We have an even state and an odd state
  - For each bit, move right, switch states if 1
  - Halt when you get to end of input
- How long will MAJ take on a Turing Machine?
  - Find a zero, cross it off
  - Go to beginning
  - Find a one, cross it off
  - Go to beginning
  - Halt when no more 0s or no more 1s

More time gives more functions

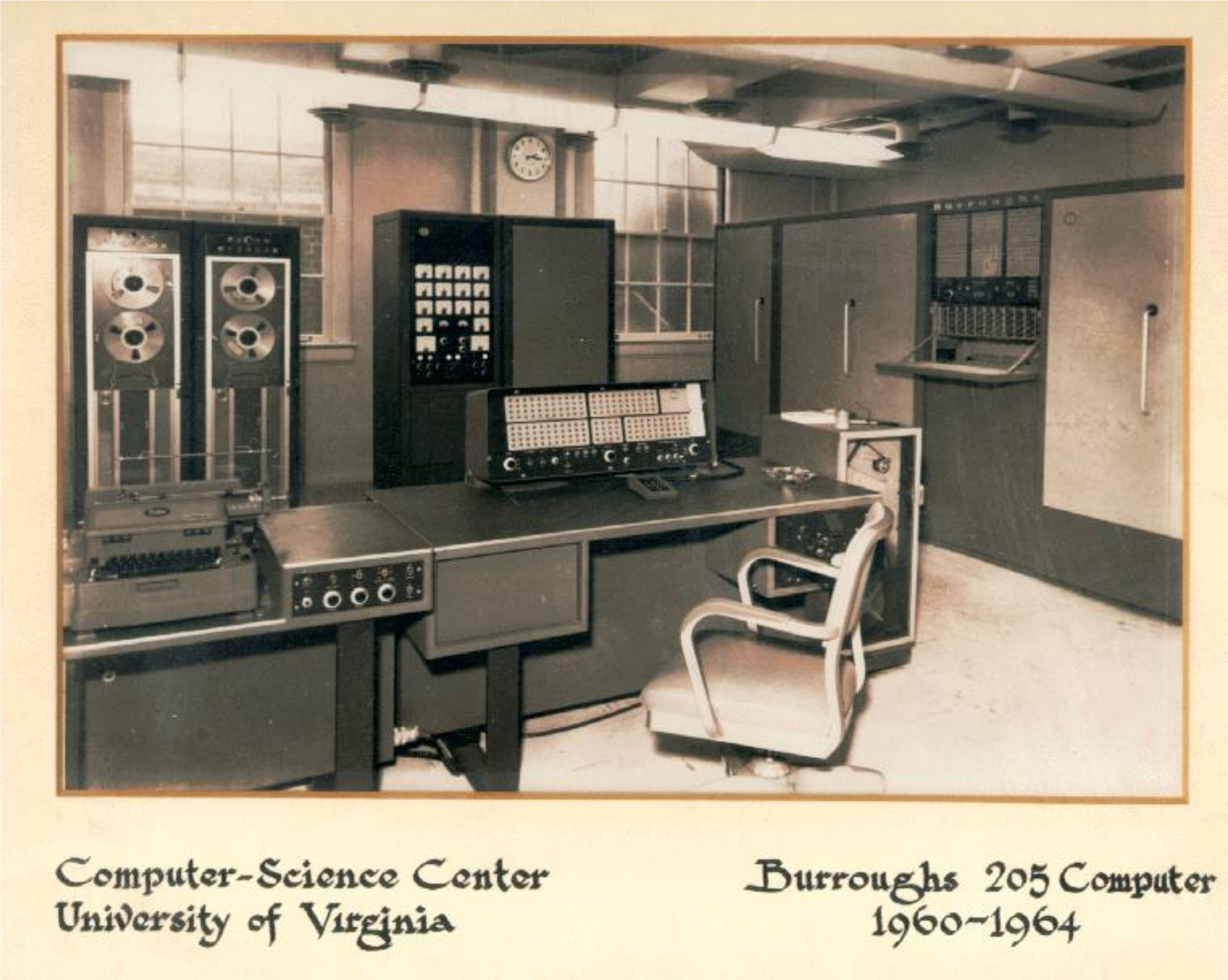


Different computing Models may have Different Running Times

- So far: a TM uses a tape. Can only visit a neighboring cell from the current one.

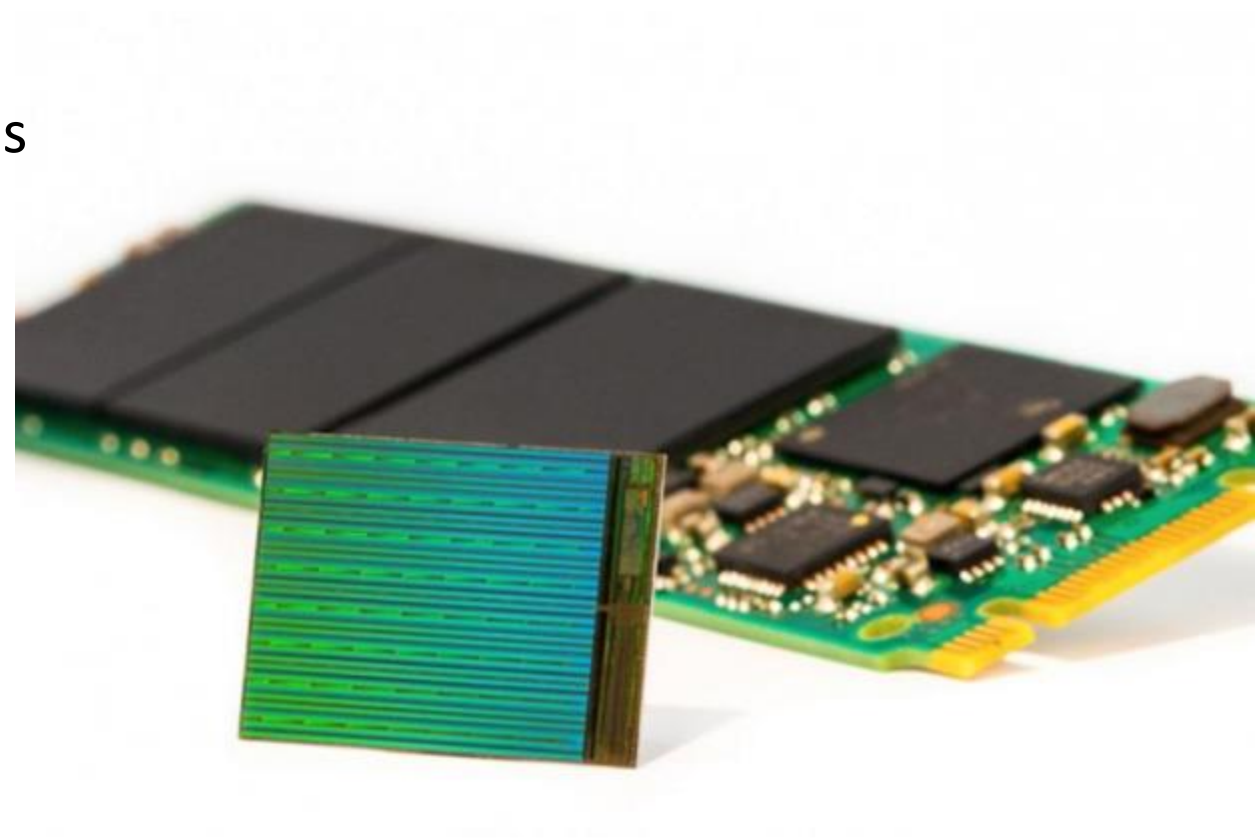
1960s

A tape was probably a reasonable memory model



Today

Can look up two locations without visiting all locations between.
“Random” access (RAM)



RAM Machine

- We can go directly to a certain index in the tape

To transition:

1. Have a second tape to keep track of current location (increment each time we move right, decrement for left)
2. Have a third tape to record the target location
3. Move until the two tapes match
    1. Maybe we need another tape to do this?

(details not important, but if you want them, see 7.2 in text)

Important observation: Tape-machine takes more steps than a RAM machine (if a RAM-TM computes f in $T(n)$ time, a tape TM can compute f in $\left(T(n)\right)^4$ time, see theorem 12.5 for details)

Finding Running Times

- We will find running times for the following:
  - Shortest Path in a graph
  - Longest Path in a graph
  - 3SAT
  - 2SAT

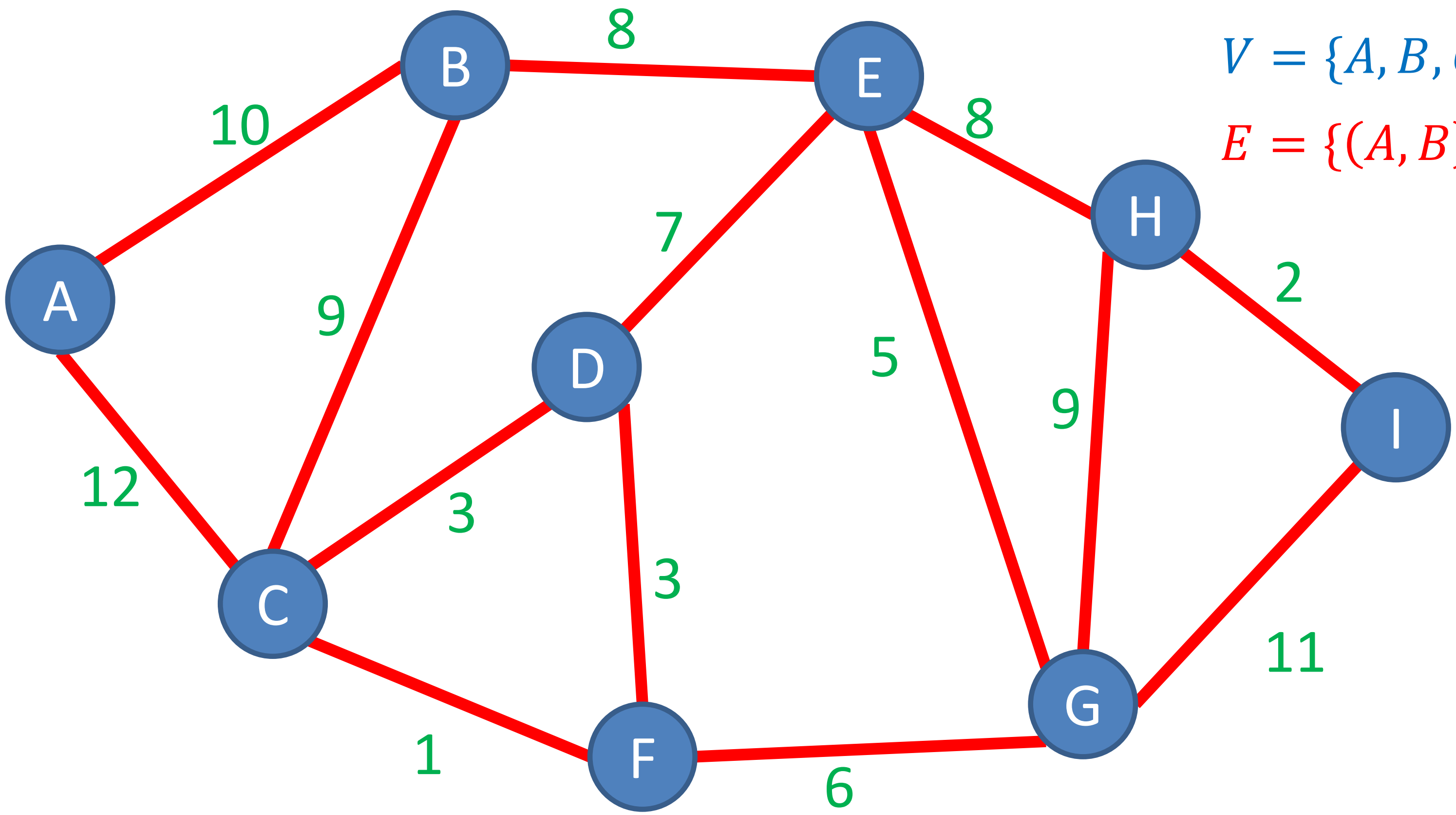
Graphs

Definition: $G = (V, E)$

Vertices/Nodes

Edges

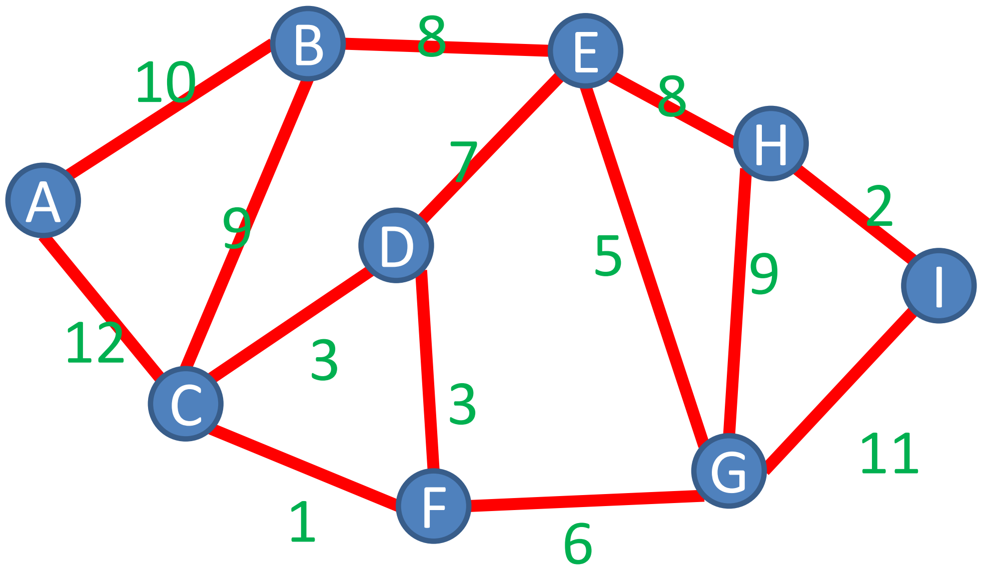
$w(e)$ = weight of edge e



$V = \{A, B, C, D, E, F, G, H, I\}$

$E = \{(A, B), (A, C), (B, C), \dots\}$

Adjacency List Representation



A	B	C		
B	A	C	E	
C	A	B	D	F
D	C	E	F	
E	B	D	G	H
F	C	D	G	
G	E	F	H	I
H	E	G	I	
I	G	H		

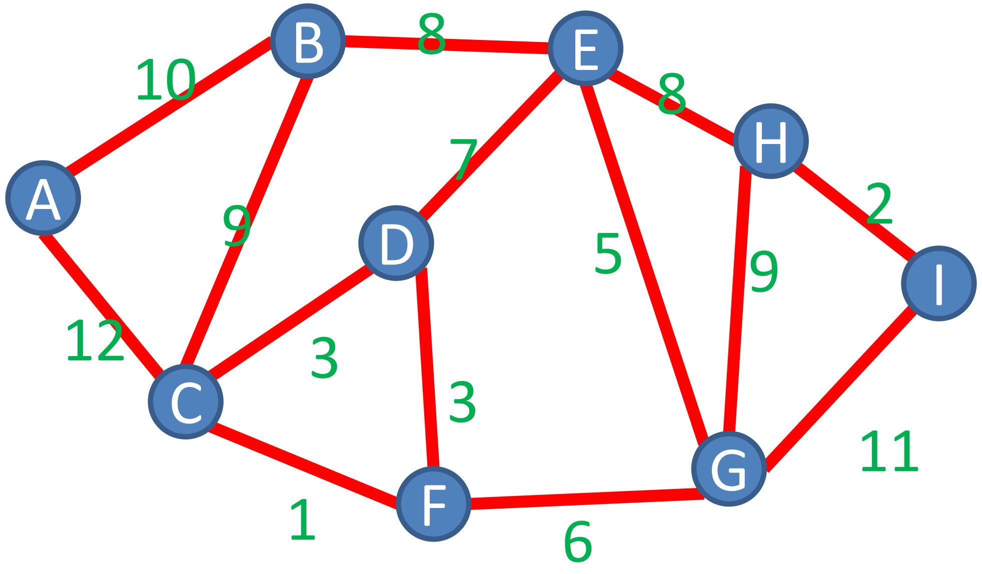
<u>Tradeoffs</u>

Space: $V + E$

Time to list neighbors: $Degree(A)$

Time to check edge (A, B): $Degree(A)$

Adjacency Matrix Representation



	A	B	C	D	E	F	G	H	I
A		1	1						
B	1		1		1				
C	1	1		1		1			
D			1		1	1			
E		1		1			1	1	
F			1	1			1		
G					1	1		1	1
H					1		1		1
I							1	1	

<u>Tradeoffs</u>

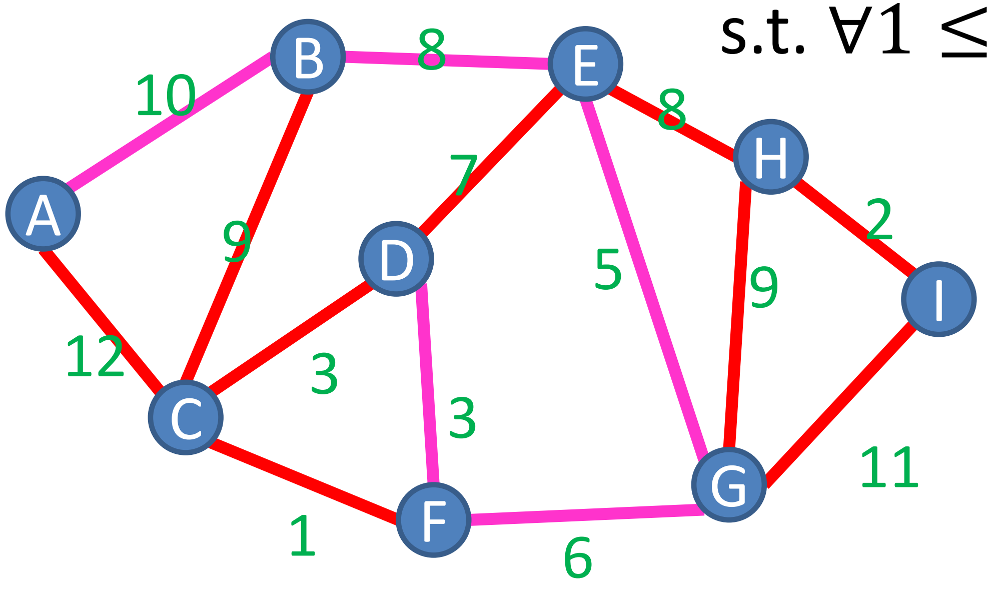
Space: V^2

Time to list neighbors: V

Time to check edge (A, B):$O(1)$

Definition: Path

A sequence of nodes $(v_1, v_2, \dots, v_k)$
s.t. $\forall 1 \leq i \leq k-1, (v_i, v_{i+1}) \in E$

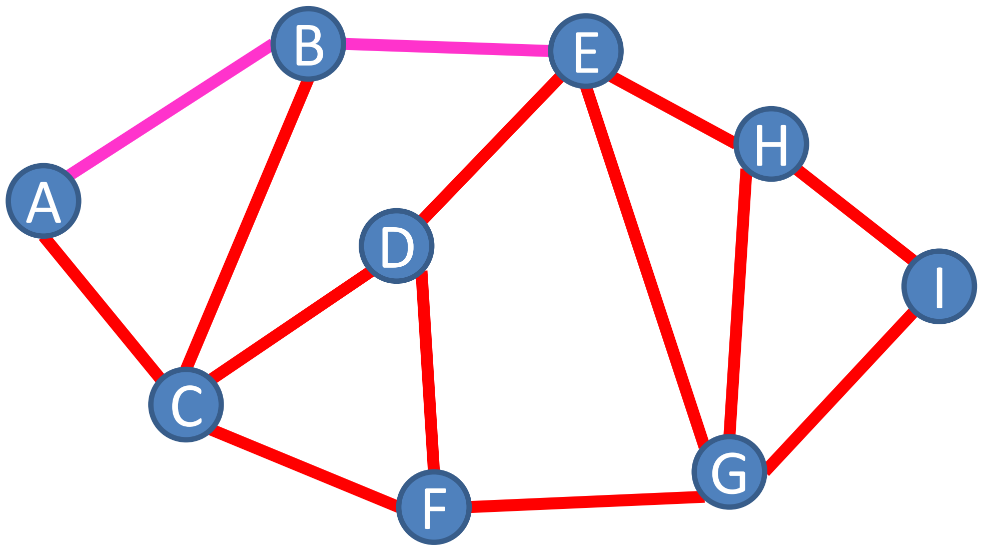


<u>Simple Path:</u>
A path in which each node appears at most once

<u>Cycle:</u>
A path of >2 nodes in which $v_1 = v_k$

Shortest Path

- Given an unweighted graph, start node s and an end node t, how long is shortest path from s to t?



Shortest path from A to E has length 2

Breadth First Search

Find a path from s to t

Keep a queue Q of nodes

$hops = 0$

$Q.enqueue((s, hops))$

While Q is not empty and $v \mathrel{!}= t$:

$\quad v, hops = Q.dequeue()$

$\quad$ for each “unvisited” $u \in V$ s.t. $(v, u) \in E$:

$\quad\quad Q.enqueue((u, hops + 1))$

Running time: $O(|V| + |E|)$

Longest Path

- Given a start node s and an end node t, how long is longest path from s to t?

Longest Path

Enumerate all possible sequences of nodes
check if it's a path
print the length of the longest one

Running time: $n!$