Circuits

Lecture 11
Uniform Circuit Complexity

Recall

Recall

- Non-uniform complexity

Recall

- Non-uniform complexity
  - P/1 $\not\subseteq$ Decidable

Recall

- Non-uniform complexity
  - $P/1 \not\subseteq$ Decidable
  - $NP \subseteq P/\log \Rightarrow NP = P$

Recall

- Non-uniform complexity
  - $P/1 \not\subseteq$ Decidable
  - $NP \subseteq P/\log \Rightarrow NP = P$
  - $NP \subseteq P/\text{poly} \Rightarrow PH = \Sigma_2^P$

Recall

- Non-uniform complexity
  - $P/1 \not\subseteq$ Decidable
  - $NP \subseteq P/\log \Rightarrow NP = P$
  - $NP \subseteq P/\text{poly} \Rightarrow PH = \Sigma_2^P$
- Circuit Complexity

Recall

- Non-uniform complexity
  - $P/1 \not\subseteq$ Decidable
  - $NP \subseteq P/log \Rightarrow NP = P$
  - $NP \subseteq P/poly \Rightarrow PH = \Sigma_2^P$
- Circuit Complexity
  - SIZE(poly) = P/poly

Recall

- Non-uniform complexity
  - $P/1 \not\subseteq$ Decidable
  - $NP \subseteq P/log \Rightarrow NP = P$
  - $NP \subseteq P/poly \Rightarrow PH = \Sigma_2^P$
- Circuit Complexity
  - SIZE(poly) = P/poly
  - SIZE-hierarchy

Recall

- Non-uniform complexity
  - $P/1 \not\subseteq$ Decidable
  - $NP \subseteq P/\log \Rightarrow NP = P$
  - $NP \subseteq P/poly \Rightarrow PH = \Sigma_2^P$
- Circuit Complexity
  - SIZE(poly) = P/poly
  - SIZE-hierarchy
    - $SIZE(T') \subsetneq SIZE(T)$ if $T=\Omega(t2^t)$ and $T'=O(2^t/t)$

Recall

- Non-uniform complexity
  - $P/1 \not\subseteq$ Decidable
  - $NP \subseteq P/\log \Rightarrow NP = P$
  - $NP \subseteq P/poly \Rightarrow PH = \Sigma_2^P$
- Circuit Complexity
  - SIZE(poly) = P/poly
  - SIZE-hierarchy
    - $SIZE(T') \subsetneq SIZE(T)$ if $T=\Omega(t2^t)$ and $T'=O(2^t/t)$
    - Most functions on t bits (that ignore last n-t bits) are in SIZE(T) but not in SIZE(T')

Uniform Circuits

Uniform Circuits

- Uniform circuit family: constructed by a TM

Uniform Circuits

- Uniform circuit family: constructed by a TM
  - Undecidable languages are undecidable for these circuits families

Uniform Circuits

- Uniform circuit family: constructed by a TM
  - Undecidable languages are undecidable for these circuits families
  - Can relate their complexity classes to classes defined using TMs

Uniform Circuits

- Uniform circuit family: constructed by a TM
  - Undecidable languages are undecidable for these circuits families
  - Can relate their complexity classes to classes defined using TMs
- Logspace-uniform:

Uniform Circuits

- Uniform circuit family: constructed by a TM
  - Undecidable languages are undecidable for these circuits families
  - Can relate their complexity classes to classes defined using TMs
- Logspace-uniform:
  - An O(log n) space TM can compute the circuit

NC^i and AC^i

NC^i and AC^i

- NC^i: class of languages decided by bounded fan-in logspace-uniform circuits of polynomial size and depth $O(\log^i n)$

NC^i and AC^i

- NC^i: class of languages decided by bounded fan-in logspace-uniform circuits of polynomial size and depth $O(\log^i n)$
  - AC^i: Similar, but unbounded fan-in circuits

NC^i and AC^i

- NC^i: class of languages decided by bounded fan-in logspace-uniform circuits of polynomial size and depth $O(\log^i n)$
  - AC^i: Similar, but unbounded fan-in circuits
- NC^0 and AC^0: constant depth circuits

NC^i and AC^i

- NC^i: class of languages decided by bounded fan-in logspace-uniform circuits of polynomial size and depth $O(\log^i n)$
  - AC^i: Similar, but unbounded fan-in circuits
- NC^0 and AC^0: constant depth circuits
  - NC^0 output depends on only a constant number of input bits

NC^i and AC^i

- NC^i: class of languages decided by bounded fan-in logspace-uniform circuits of polynomial size and depth $O(\log^i n)$
  - AC^i: Similar, but unbounded fan-in circuits
- NC^0 and AC^0: constant depth circuits
  - NC^0 output depends on only a constant number of input bits
  - $NC^0 \subsetneq AC^0$: Consider L = {1,11,111,...}

NC and AC

NC and AC

- NC = $\cup_{i>0}$ NC^i. Similarly AC.

NC and AC

- $NC = \cup_{i>0} NC^i$. Similarly AC.
- $NC^i \subseteq AC^i \subseteq NC^{i+1}$

NC and AC

- $NC = \cup_{i>0} NC^i$. Similarly AC.
- $NC^i \subseteq AC^i \subseteq NC^{i+1}$
  - Clearly $NC^i \subseteq AC^i$

NC and AC

- $NC = \cup_{i>0} NC^i$. Similarly AC.
- $NC^i \subseteq AC^i \subseteq NC^{i+1}$
  - Clearly $NC^i \subseteq AC^i$
  - $AC^i \subseteq NC^{i+1}$ because polynomial fan-in can be reduced to constant fan-in by using a log depth tree

NC and AC

- NC = $\cup_{i>0}$ NC^i. Similarly AC.
- $NC^i \subseteq AC^i \subseteq NC^{i+1}$
  - Clearly $NC^i \subseteq AC^i$
  - $AC^i \subseteq NC^{i+1}$ because polynomial fan-in can be reduced to constant fan-in by using a log depth tree
- So NC = AC

$NC \subseteq P$

NC $\subseteq$ P

- Generate circuit of the right input size and evaluate on input

NC ⊆ P

- Generate circuit of the right input size and evaluate on input
- Generating the circuit

NC ⊆ P

- Generate circuit of the right input size and evaluate on input
- Generating the circuit
  - in logspace, so poly time; also circuit size is poly

NC $\subseteq$ P

- Generate circuit of the right input size and evaluate on input
- Generating the circuit
  - in logspace, so poly time; also circuit size is poly
- Evaluating the gates

NC $\subseteq$ P

- Generate circuit of the right input size and evaluate on input
- Generating the circuit
  - in logspace, so poly time; also circuit size is poly
- Evaluating the gates
  - Poly(n) gates

NC ⊆ P

- Generate circuit of the right input size and evaluate on input
- Generating the circuit
  - in logspace, so poly time; also circuit size is poly
- Evaluating the gates
  - Poly(n) gates
  - Per gate takes O(1) time + time to look up output values of (already evaluated) gates

NC ⊆ P

- Generate circuit of the right input size and evaluate on input
- Generating the circuit
  - in logspace, so poly time; also circuit size is poly
- Evaluating the gates
  - Poly(n) gates
  - Per gate takes O(1) time + time to look up output values of (already evaluated) gates
- Open problem: Is NC = P?

Motivation for NC

Motivation for NC

- Fast parallel computation is (loosely) modeled as having poly many processors and taking poly-log time

Motivation for NC

- Fast parallel computation is (loosely) modeled as having poly many processors and taking poly-log time
  - Corresponds to NC (How?)

Motivation for NC

- Fast parallel computation is (loosely) modeled as having poly many processors and taking poly-log time
  - Corresponds to NC (How?)
  - Depth translates to time

Motivation for NC

- Fast parallel computation is (loosely) modeled as having poly many processors and taking poly-log time
  - Corresponds to NC (How?)
  - Depth translates to time
  - Total "work" is size of the circuit

An example

An example

- PARITY in NC^1

An example

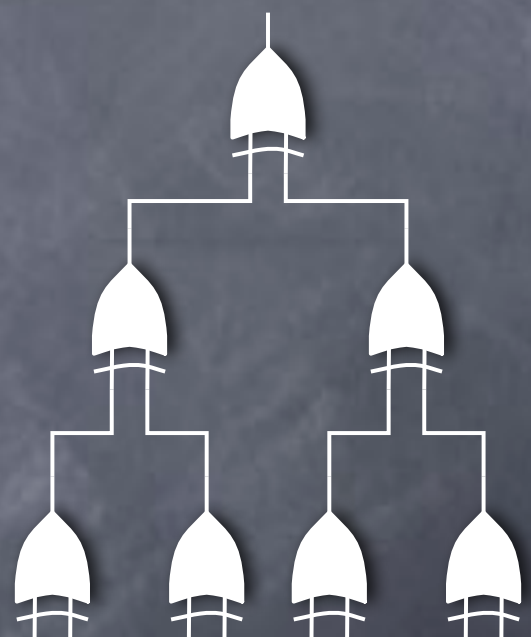
- PARITY in NC^1
  - PARITY = { x | x has odd number of 1s }

An example

- PARITY in NC^1
  - PARITY = { x | x has odd number of 1s }
  - Circuit should evaluate $x_1 \oplus x_2 \oplus \ldots \oplus x_n$

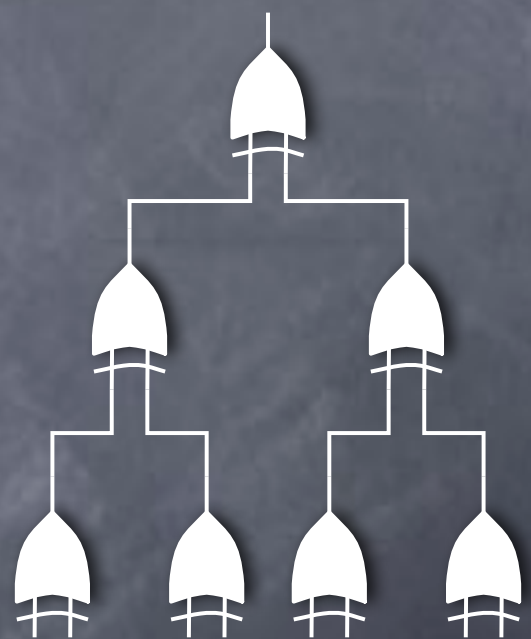
An example

- PARITY in NC^1
  - PARITY = { x | x has odd number of 1s }
  - Circuit should evaluate $x_1 \oplus x_2 \oplus \ldots \oplus x_n$
    - Tree of n-1 XOR gates: log n deep



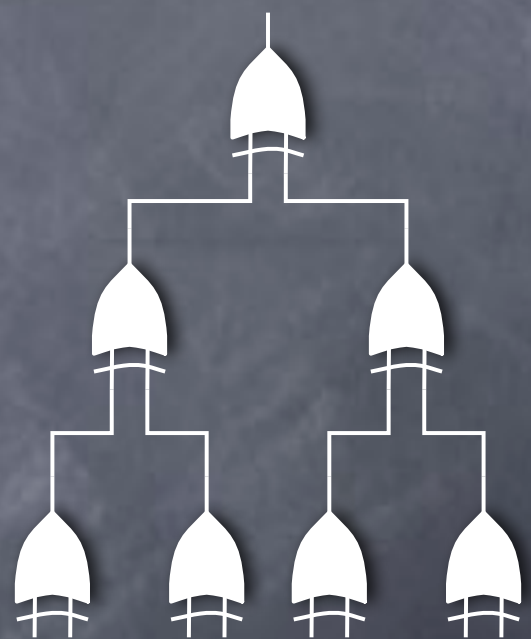
An example

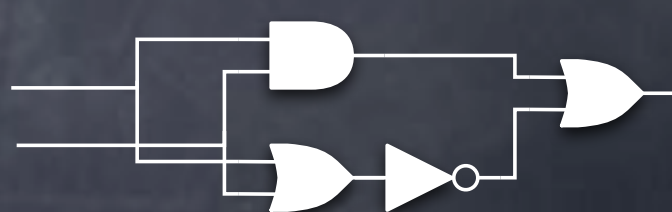
- PARITY in NC^1
  - PARITY = { x | x has odd number of 1s }
  - Circuit should evaluate $x_1 \oplus x_2 \oplus \ldots \oplus x_n$
    - Tree of n-1 XOR gates: log n deep
    - Each XOR gate implemented in depth 3



An example

- PARITY in NC^1
  - PARITY = { x | x has odd number of 1s }
  - Circuit should evaluate $x_1 \oplus x_2 \oplus \ldots \oplus x_n$
    - Tree of n-1 XOR gates: log n deep
    - Each XOR gate implemented in depth 3





Another example

Another example

- PATH $\in AC^1$

Another example

- PATH $\in AC^1$
  - "Boolean" Matrix Multiplication

Another example

- PATH $\in AC^1$
  - "Boolean" Matrix Multiplication
    - $Z=XY$: $z_{ij} = \vee_{k=1..n} (x_{ik} \wedge y_{kj})$

Another example

- PATH $\in AC^1$
  - "Boolean" Matrix Multiplication
    - $Z=XY$: $z_{ij} = \vee_{k=1..n} (x_{ik} \wedge y_{kj})$
    - AC^0 circuit (OR gate with fan-in n, binary AND gates)

Another example

- PATH $\in AC^1$
  - "Boolean" Matrix Multiplication
    - $Z=XY$: $z_{ij} = \vee_{k=1..n} (x_{ik} \wedge y_{kj})$
    - AC^0 circuit (OR gate with fan-in n, binary AND gates)
  - If X adjacency matrix (with self-loops), $X^t_{ij}=1$ iff path from i to j of length t or less

Another example

- PATH $\in AC^1$
  - "Boolean" Matrix Multiplication
    - $Z=XY$: $z_{ij} = \vee_{k=1..n} (x_{ik} \wedge y_{kj})$
    - AC^0 circuit (OR gate with fan-in n, binary AND gates)
  - If X adjacency matrix (with self-loops), $X^t_{ij}=1$ iff path from i to j of length t or less
    - X^m_{ij} for $m \geq n$ is the transitive closure

Another example

- PATH $\in AC^1$
  - "Boolean" Matrix Multiplication
    - $Z=XY$: $z_{ij} = \vee_{k=1..n} (x_{ik} \wedge y_{kj})$
    - AC^0 circuit (OR gate with fan-in n, binary AND gates)
  - If X adjacency matrix (with self-loops), $X^t_{ij}=1$ iff path from i to j of length t or less
    - X^m_{ij} for $m \geq n$ is the transitive closure
    - O(log n) matrix multiplications to compute X^n_{ij}

Another example

- PATH $\in AC^1$
  - "Boolean" Matrix Multiplication
    - $Z=XY$: $z_{ij} = \vee_{k=1..n} (x_{ik} \wedge y_{kj})$
    - AC^0 circuit (OR gate with fan-in n, binary AND gates)
  - If X adjacency matrix (with self-loops), $X^t_{ij}=1$ iff path from i to j of length t or less
    - X^m_{ij} for $m \geq n$ is the transitive closure
    - O(log n) matrix multiplications to compute X^n_{ij}
    - Total depth O(log n)

$NC^1 \subseteq L$

$NC^1 \subseteq L$

- Generate circuit (implicitly) and evaluate

$NC^1 \subseteq L$

- Generate circuit (implicitly) and evaluate
  - cf. $NC \subseteq P$. But now, to conserve space, a recursive evaluation (rather than bottom-up).

$NC^1 \subseteq L$

- Generate circuit (implicitly) and evaluate
  - cf. $NC \subseteq P$. But now, to conserve space, a recursive evaluation (rather than bottom-up).
  - For each gate, recursively evaluate each input wire

$NC^1 \subseteq L$

- Generate circuit (implicitly) and evaluate
  - cf. NC ⊆ P. But now, to conserve space, a recursive evaluation (rather than bottom-up).
  - For each gate, recursively evaluate each input wire
    - Storage: A path to the current node, from the output node: since bounded fan-in, takes O(1) bits per node; since logspace uniform that is sufficient to compute the node id in logspace

$NC^1 \subseteq L$

- Generate circuit (implicitly) and evaluate
  - cf. NC ⊆ P. But now, to conserve space, a recursive evaluation (rather than bottom-up).
  - For each gate, recursively evaluate each input wire
    - Storage: A path to the current node, from the output node: since bounded fan-in, takes O(1) bits per node; since logspace uniform that is sufficient to compute the node id in logspace
      - And at each node along the path, the input wire values evaluated results so far (again O(1) bits per node)

$NC^1 \subseteq L$

- Generate circuit (implicitly) and evaluate
  - cf. NC ⊆ P. But now, to conserve space, a recursive evaluation (rather than bottom-up).
  - For each gate, recursively evaluate each input wire
    - Storage: A path to the current node, from the output node: since bounded fan-in, takes O(1) bits per node; since logspace uniform that is sufficient to compute the node id in logspace
      - And at each node along the path, the input wire values evaluated results so far (again O(1) bits per node)
    - Length of path = depth of circuit = O(log n)

$NL \subseteq AC^1$

$NL \subseteq AC^1$

- Recall $PATH \in AC^1$

$NL \subseteq AC^1$

- Recall PATH $\in AC^1$
- Also recall PATH is NL-complete

$NL \subseteq AC^1$

- Recall PATH $\in AC^1$
- Also recall PATH is NL-complete
  - with respect to log-space reductions

$NL \subseteq AC^1$

- Recall PATH $\in AC^1$
- Also recall PATH is NL-complete
  - with respect to log-space reductions
  - in fact, with respect to NC^1 reductions

$NL \subseteq AC^1$

- Recall PATH $\in AC^1$
- Also recall PATH is NL-complete
  - with respect to log-space reductions
  - in fact, with respect to NC^1 reductions
    - For NL machine M, can build (in log-space) an NC^1 circuit which on input x, outputs $(i,j)^{th}$ entry of the adjacency matrix of configuration graph of M(x). (Configuration 0, the start configuration depends on x.)

$NL \subseteq AC^1$

- Recall PATH $\in AC^1$
- Also recall PATH is NL-complete
  - with respect to log-space reductions
  - in fact, with respect to NC^1 reductions
    - For NL machine M, can build (in log-space) an NC^1 circuit which on input x, outputs $(i,j)^{th}$ entry of the adjacency matrix of configuration graph of M(x). (Configuration 0, the start configuration depends on x.)
  - Combining the NC^1 circuits for reduction and the AC^1 circuit for PATH, we get an AC^1 circuit

Summary: NC^i and AC^i

Summary: NC^i and AC^i

- $NC^i \subseteq AC^i \subseteq NC^{i+1} \subseteq NC = AC \subseteq P$

Summary: NC^i and AC^i

- $NC^i \subseteq AC^i \subseteq NC^{i+1} \subseteq NC = AC \subseteq P$
- $NC^0 \subsetneq AC^0 \subsetneq NC^1 \subseteq L \subseteq NL \subseteq AC^1$

Summary: NC^i and AC^i

- $NC^i \subseteq AC^i \subseteq NC^{i+1} \subseteq NC = AC \subseteq P$
- $NC^0 \subsetneq AC^0 \subsetneq NC^1 \subseteq L \subseteq NL \subseteq AC^1$
  - $AC^0 \subsetneq NC^1$ as PARITY $\notin AC^0$ (later)

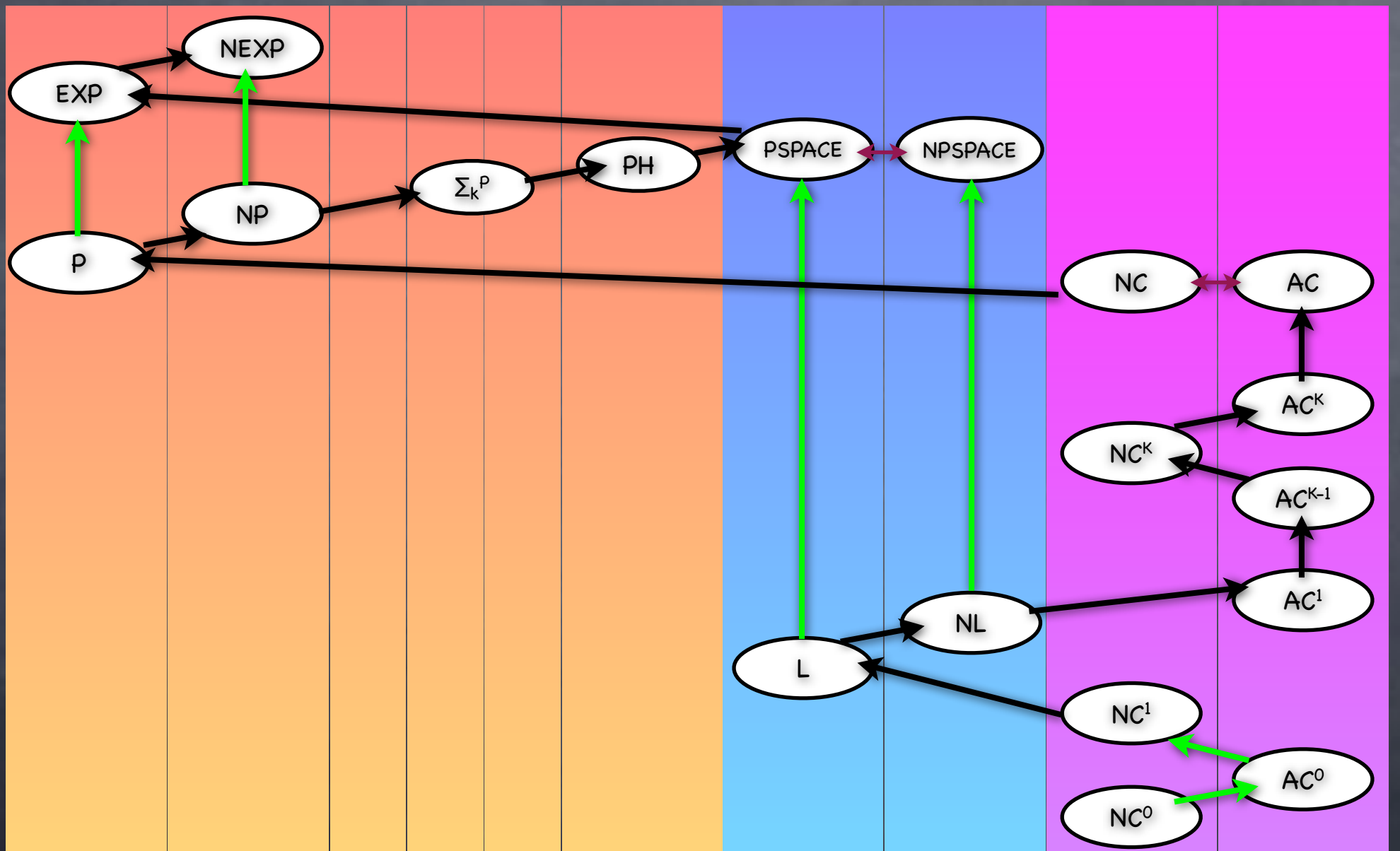
Summary: NC^i and AC^i

- $NC^i \subseteq AC^i \subseteq NC^{i+1} \subseteq NC = AC \subseteq P$
- $NC^0 \subsetneq AC^0 \subsetneq NC^1 \subseteq L \subseteq NL \subseteq AC^1$
  - $AC^0 \subsetneq NC^1$ as PARITY $\notin AC^0$ (later)
- Open: whether $NC^i \subsetneq AC^i \subsetneq NC^{i+1}$ for larger i

Summary: NC^i and AC^i

- $NC^i \subseteq AC^i \subseteq NC^{i+1} \subseteq NC = AC \subseteq P$
- $NC^0 \subsetneq AC^0 \subsetneq NC^1 \subseteq L \subseteq NL \subseteq AC^1$
  - $AC^0 \subsetneq NC^1$ as PARITY $\notin AC^0$ (later)
- Open: whether $NC^i \subsetneq AC^i \subsetneq NC^{i+1}$ for larger i
- Open: Is NC = P? (Can all polynomial time decidable languages be sped up to poly-log time using parallelization?)

Zoo


NEXP
EXP
PSPACE
NPSPACE
PH
Σ_k^P
NP
P
NC
AC
AC^k
NC^k
AC^{k-1}
AC^1
NL
L
NC^1
AC^0
NC^0

DC Uniform

DC Uniform

- Recall Uniform circuit family: circuits in the family can be generated by a TM

DC Uniform

- Recall Uniform circuit family: circuits in the family can be generated by a TM
- Suppose circuits are super-polynomially large. Cannot be logspace-uniform or P-uniform.

DC Uniform

- Recall Uniform circuit family: circuits in the family can be generated by a TM
- Suppose circuits are super-polynomially large. Cannot be logspace-uniform or P-uniform.
  - DC uniform allows exponentially large circuits

DC Uniform

- Recall Uniform circuit family: circuits in the family can be generated by a TM
- Suppose circuits are super-polynomially large. Cannot be logspace-uniform or P-uniform.
  - DC uniform allows exponentially large circuits
  - Still requires polynomial time implicit computation of the circuit

DC Uniform

- Recall Uniform circuit family: circuits in the family can be generated by a TM
- Suppose circuits are super-polynomially large. Cannot be logspace-uniform or P-uniform.
  - DC uniform allows exponentially large circuits
  - Still requires polynomial time implicit computation of the circuit
  - Coincides with EXP (Why?)

O(1) depth DC Uniform

O(1) depth DC Uniform

- Restricted to constant depth, $2^{poly(n)}$ size, unbounded fan-in DC uniform circuit families decide exactly languages in PH

O(1) depth DC Uniform

- Restricted to constant depth, $2^{poly(n)}$ size, unbounded fan-in DC uniform circuit families decide exactly languages in PH
  - Given a DC uniform circuit (w.l.o.g alternating levels of AND and OR gates, and NOT gates only at the input level) of depth k, an equivalent quantified expression with k alternations

O(1) depth DC Uniform

- Restricted to constant depth, $2^{poly(n)}$ size, unbounded fan-in DC uniform circuit families decide exactly languages in PH
  - Given a DC uniform circuit (w.l.o.g alternating levels of AND and OR gates, and NOT gates only at the input level) of depth k, an equivalent quantified expression with k alternations
  - Given a quantified expression with k alternations, an equivalent DC uniform circuit of depth k+2

O(1) depth DC Uniform

O(1) depth DC Uniform

- From circuit to quantified expression

O(1) depth DC Uniform

- From circuit to quantified expression
  - Consider game played on the circuit: adversary picks an input edge to the AND level and Alice picks an input edge to the OR level, going through levels top to bottom

O(1) depth DC Uniform

- From circuit to quantified expression
  - Consider game played on the circuit: adversary picks an input edge to the AND level and Alice picks an input edge to the OR level, going through levels top to bottom
  - Alice wins if adversary "breaks off the path" (by picking either a non-wire edge or a wire not continuing the path), or if the path terminates at literal of value 1 (w/o breaking)

O(1) depth DC Uniform

- From circuit to quantified expression
  - Consider game played on the circuit: adversary picks an input edge to the AND level and Alice picks an input edge to the OR level, going through levels top to bottom
  - Alice wins if adversary "breaks off the path" (by picking either a non-wire edge or a wire not continuing the path), or if the path terminates at literal of value 1 (w/o breaking)

Can check in poly time

O(1) depth DC Uniform

- From circuit to quantified expression
  - Consider game played on the circuit: adversary picks an input edge to the AND level and Alice picks an input edge to the OR level, going through levels top to bottom
  - Alice wins if adversary "breaks off the path" (by picking either a non-wire edge or a wire not continuing the path), or if the path terminates at literal of value 1 (w/o breaking)
  - Input accepted by the circuit iff Alice has a winning strategy (i.e., if the quantified expression is true)

Can check in poly time

O(1) depth DC Uniform

- From circuit to quantified expression
  - Consider game played on the circuit: adversary picks an input edge to the AND level and Alice picks an input edge to the OR level, going through levels top to bottom

Can check in poly time

  - Alice wins if adversary "breaks off the path" (by picking either a non-wire edge or a wire not continuing the path), or if the path terminates at literal of value 1 (w/o breaking)
  - Input accepted by the circuit iff Alice has a winning strategy (i.e., if the quantified expression is true)
  - Each edge has a polynomially long label, and quantified variables take values from the same domain. Checking if edge is a correct wire in poly time (uniformity)

O(1) depth DC Uniform

O(1) depth DC Uniform

- From quantified expression to circuit:

O(1) depth DC Uniform

- From quantified expression to circuit:
  - Circuit has depth-2 sub-circuits evaluating the poly-time condition for each possible assignment of the quantified variables.

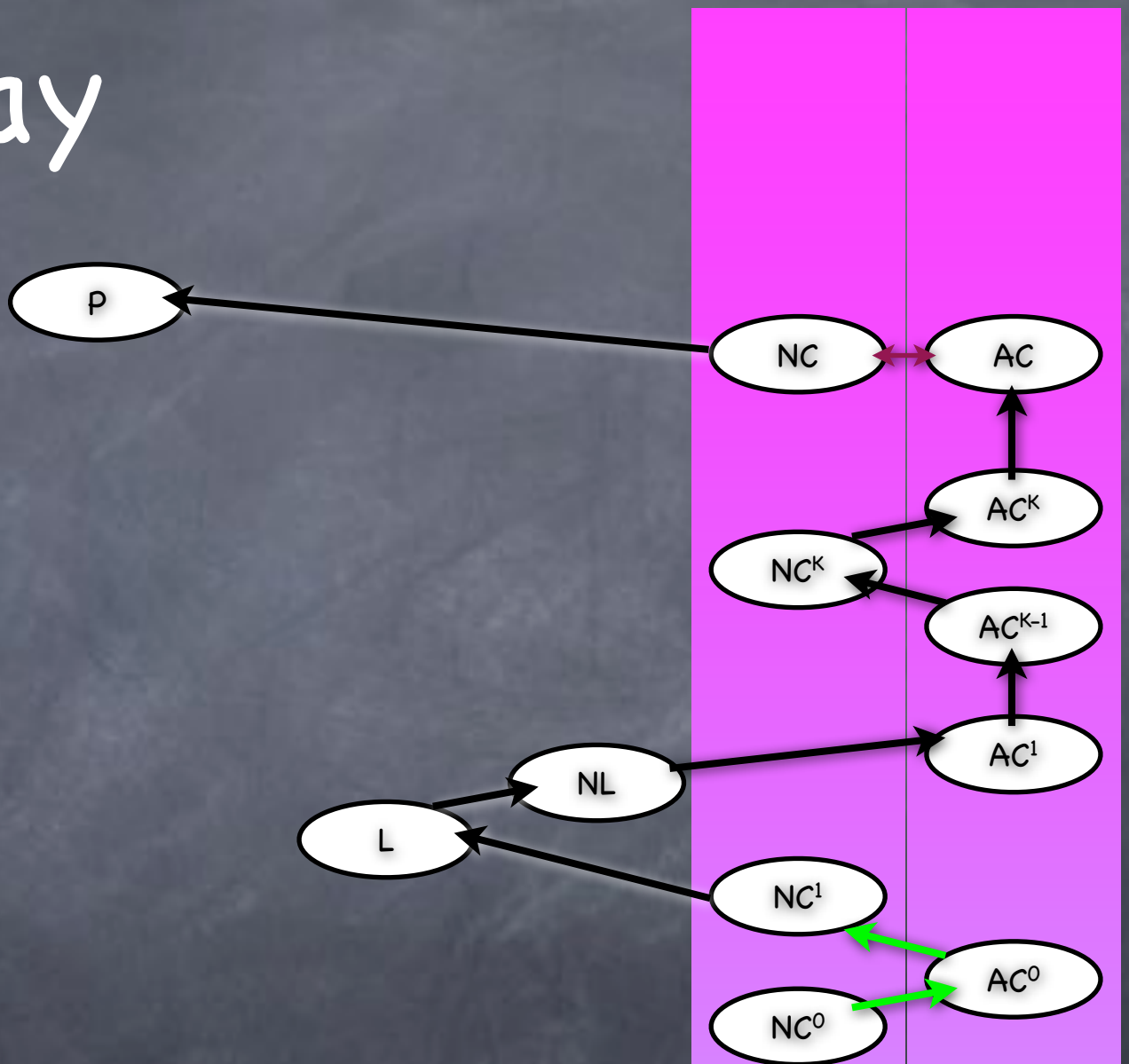
O(1) depth DC Uniform

- From quantified expression to circuit:
  - Circuit has depth-2 sub-circuits evaluating the poly-time condition for each possible assignment of the quantified variables.
  - Hang these sub-circuits at the leaves of a k-level AND-OR tree appropriately

O(1) depth DC Uniform

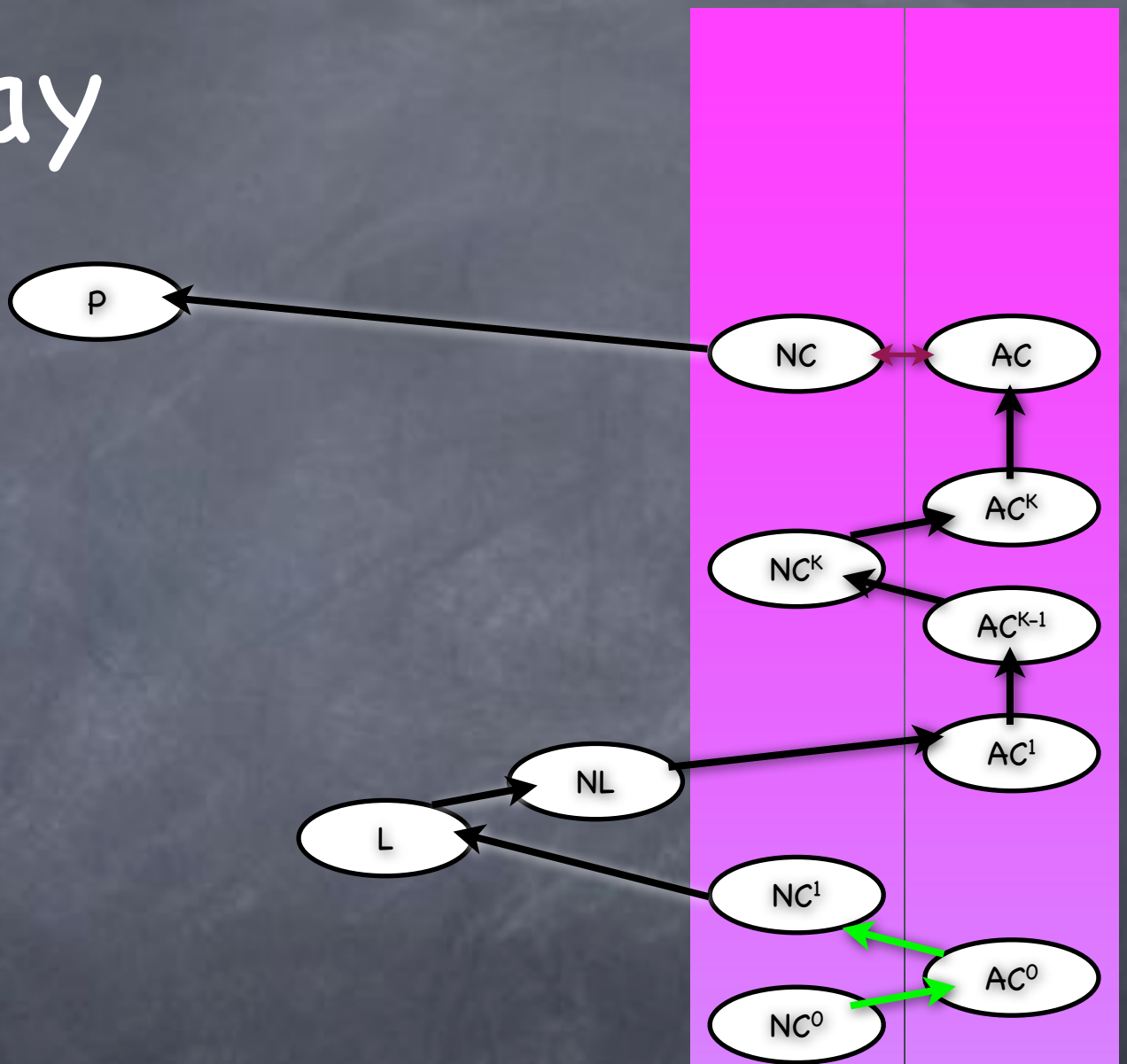
- From quantified expression to circuit:
  - Circuit has depth-2 sub-circuits evaluating the poly-time condition for each possible assignment of the quantified variables.
  - Hang these sub-circuits at the leaves of a k-level AND-OR tree appropriately
  - Circuit can be implicitly computed in polynomial time. Size $2^{O(\text{total length of variables})}$

Today



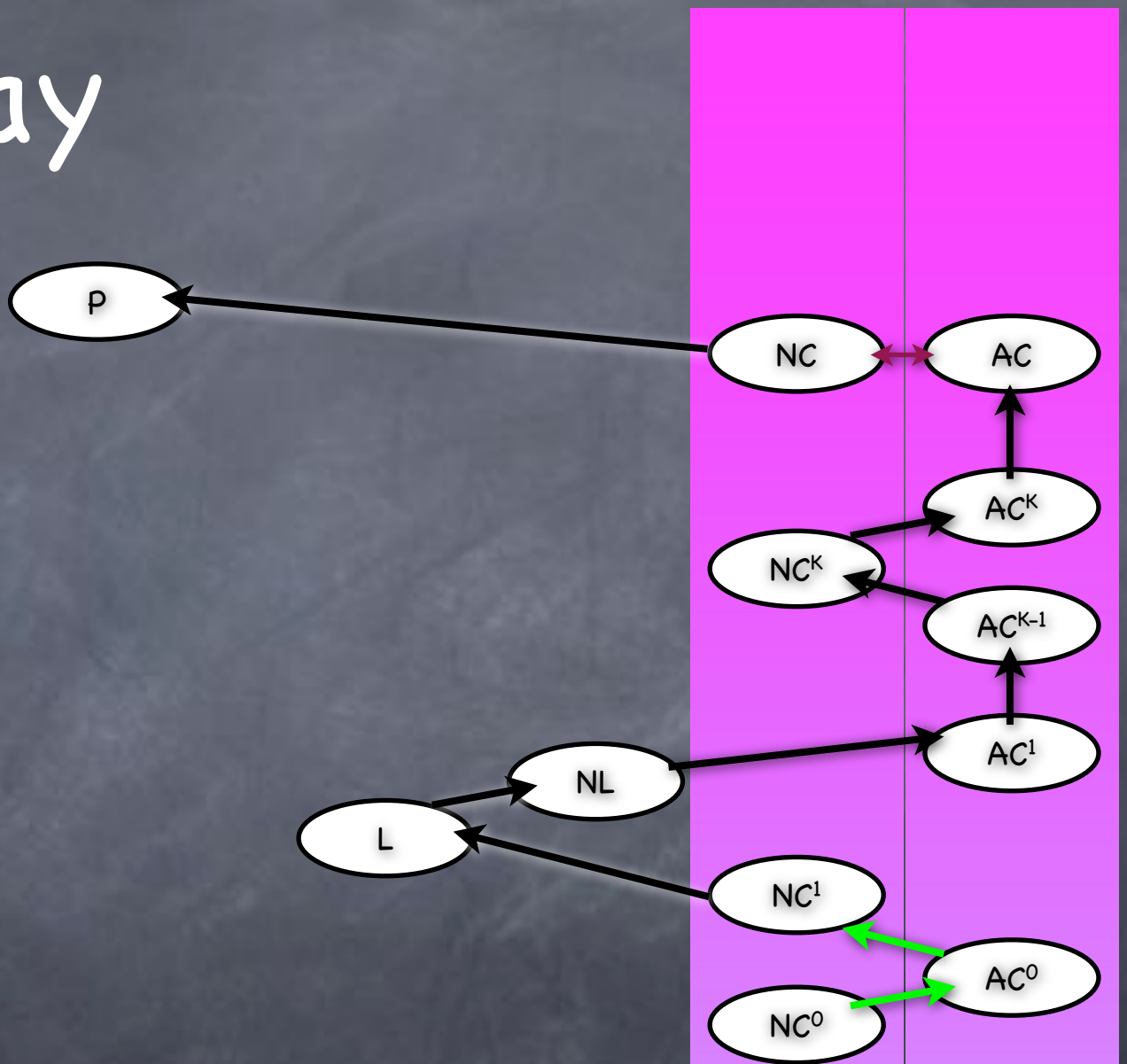
Today

- NC^i and AC^i



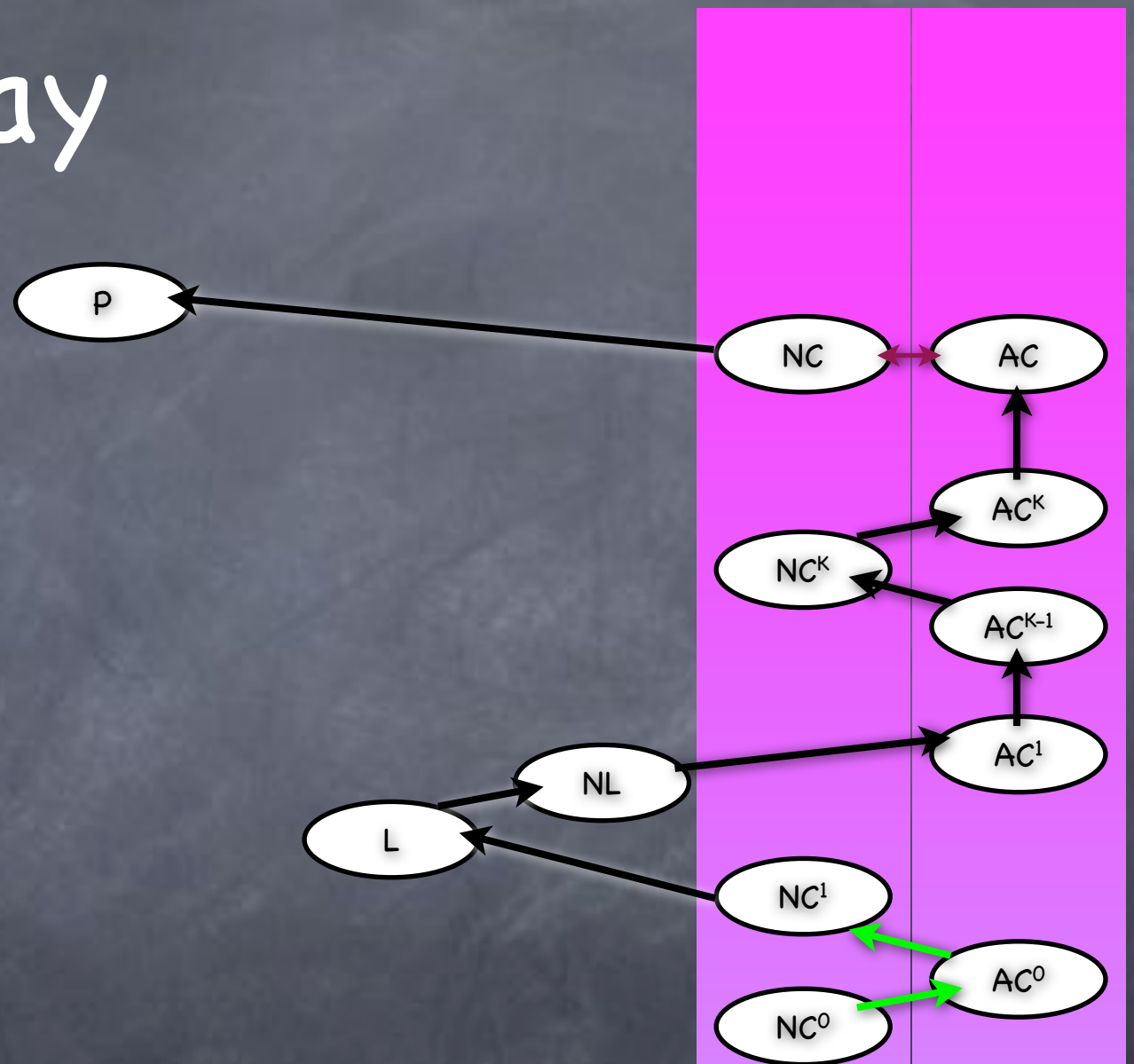
Today

- NC^i and AC^i
- DC-uniform



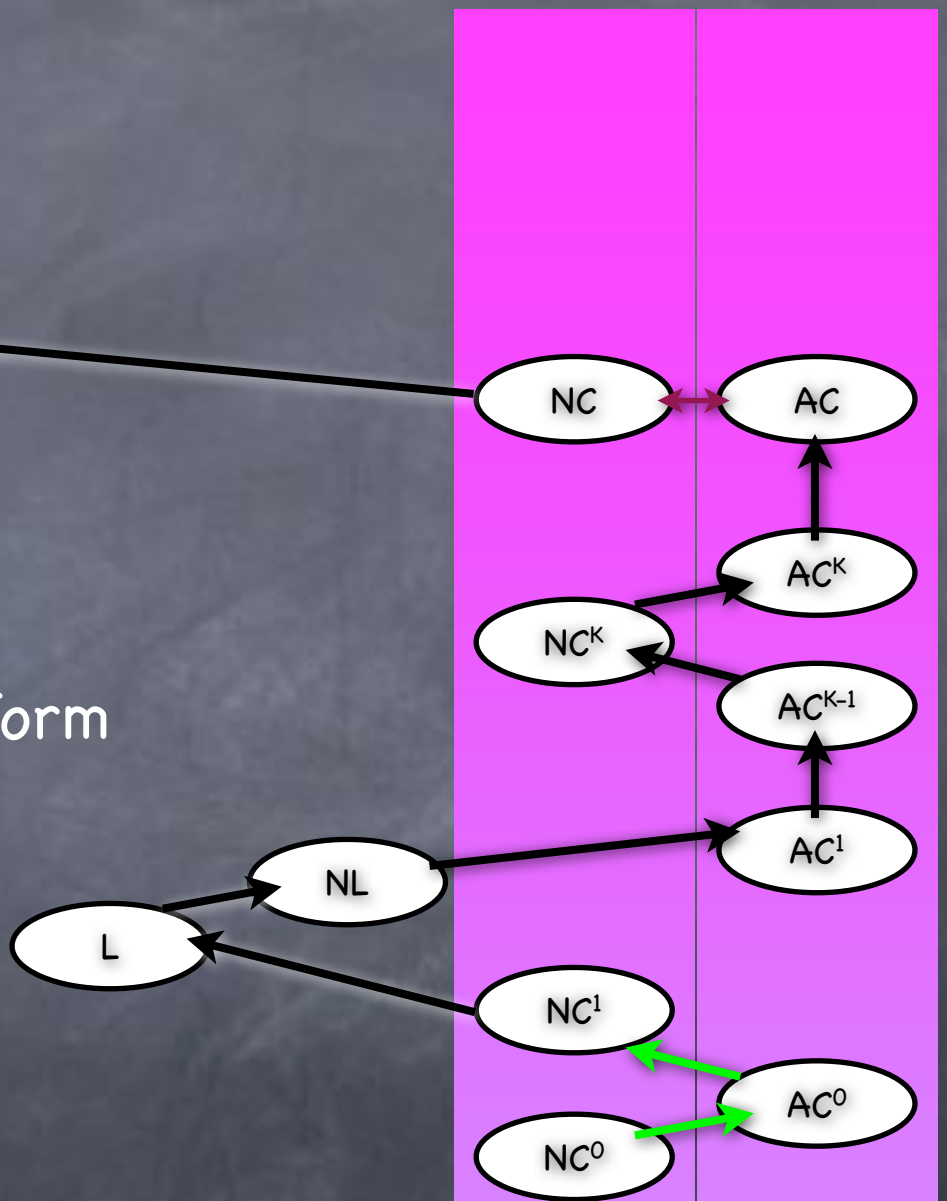
Today

- NC^i and AC^i
- DC-uniform
  - PH and EXP



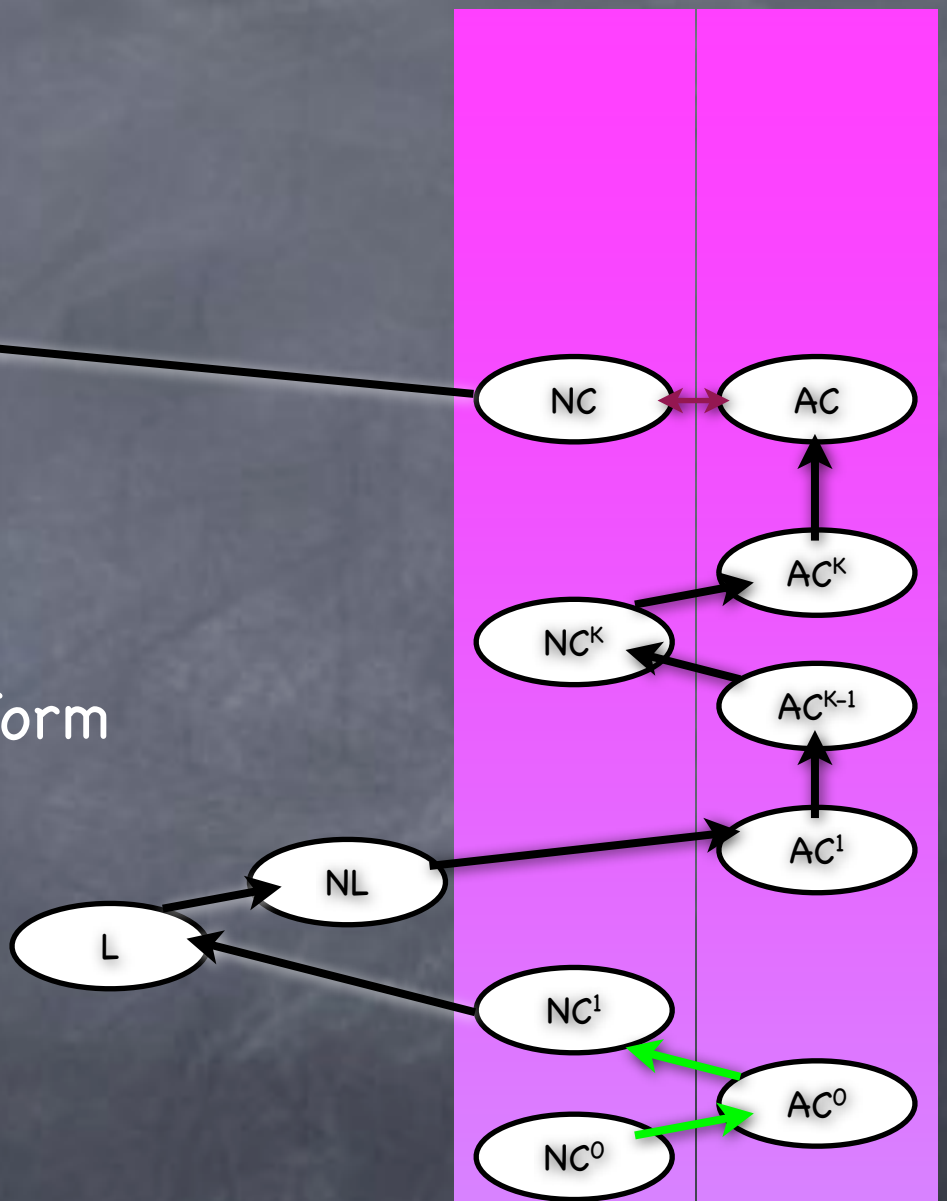
Today

- NC^i and AC^i
- DC-uniform
  - PH and EXP
- Later, more circuits and non-uniform computation (time permitting)



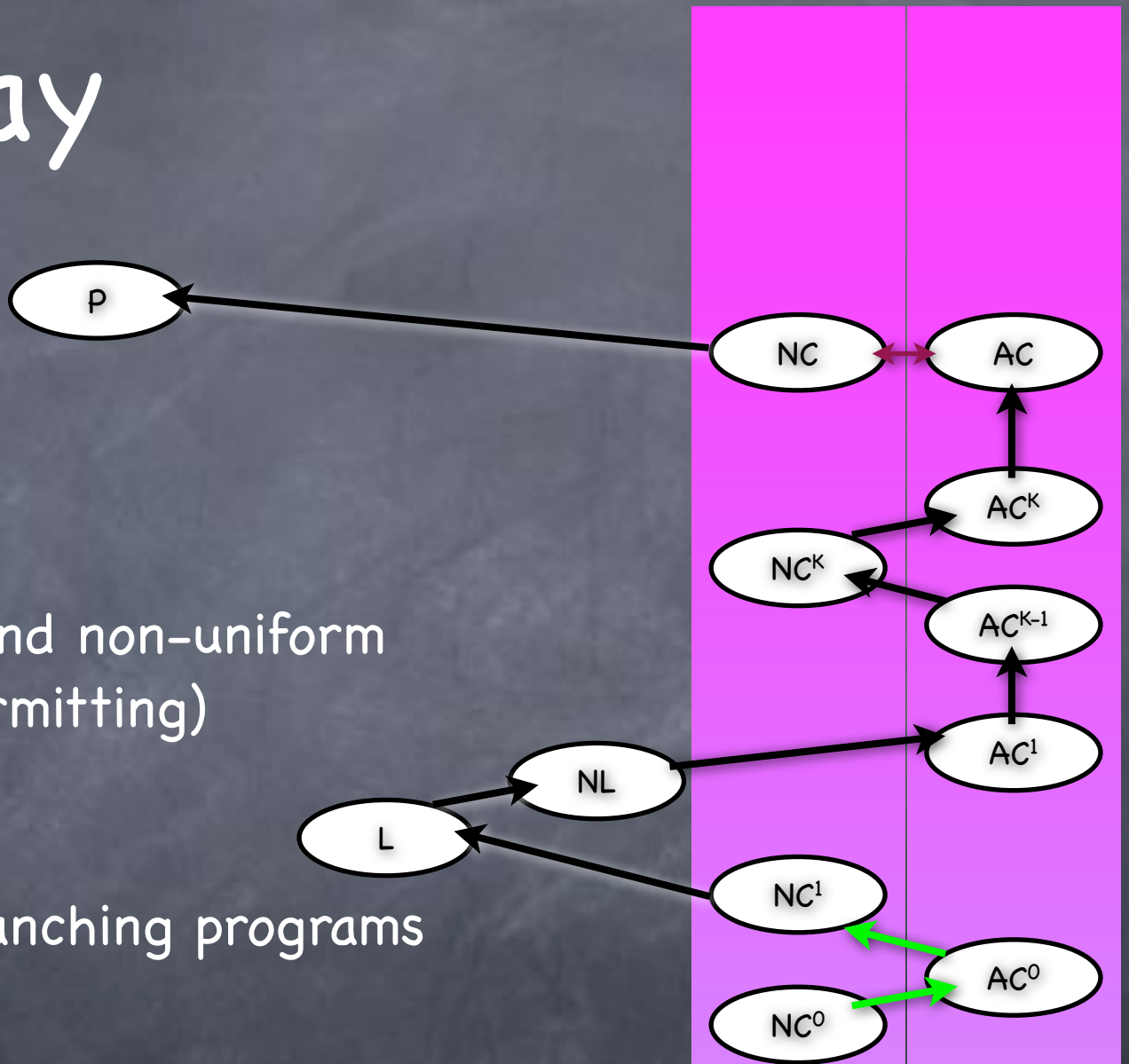
Today

- NC^i and AC^i
- DC-uniform
  - PH and EXP
- Later, more circuits and non-uniform computation (time permitting)
  - PARITY $\notin AC^0$



Today

- NC^i and AC^i
- DC-uniform
  - PH and EXP
- Later, more circuits and non-uniform computation (time permitting)
  - PARITY $\notin AC^0$
  - Decision trees, Branching programs



Today

- NC^i and AC^i
- DC-uniform
  - PH and EXP
- Later, more circuits and non-uniform computation (time permitting)
  - PARITY $\notin AC^0$
  - Decision trees, Branching programs
  - Connections between circuit lowerbounds and other complexity class separations