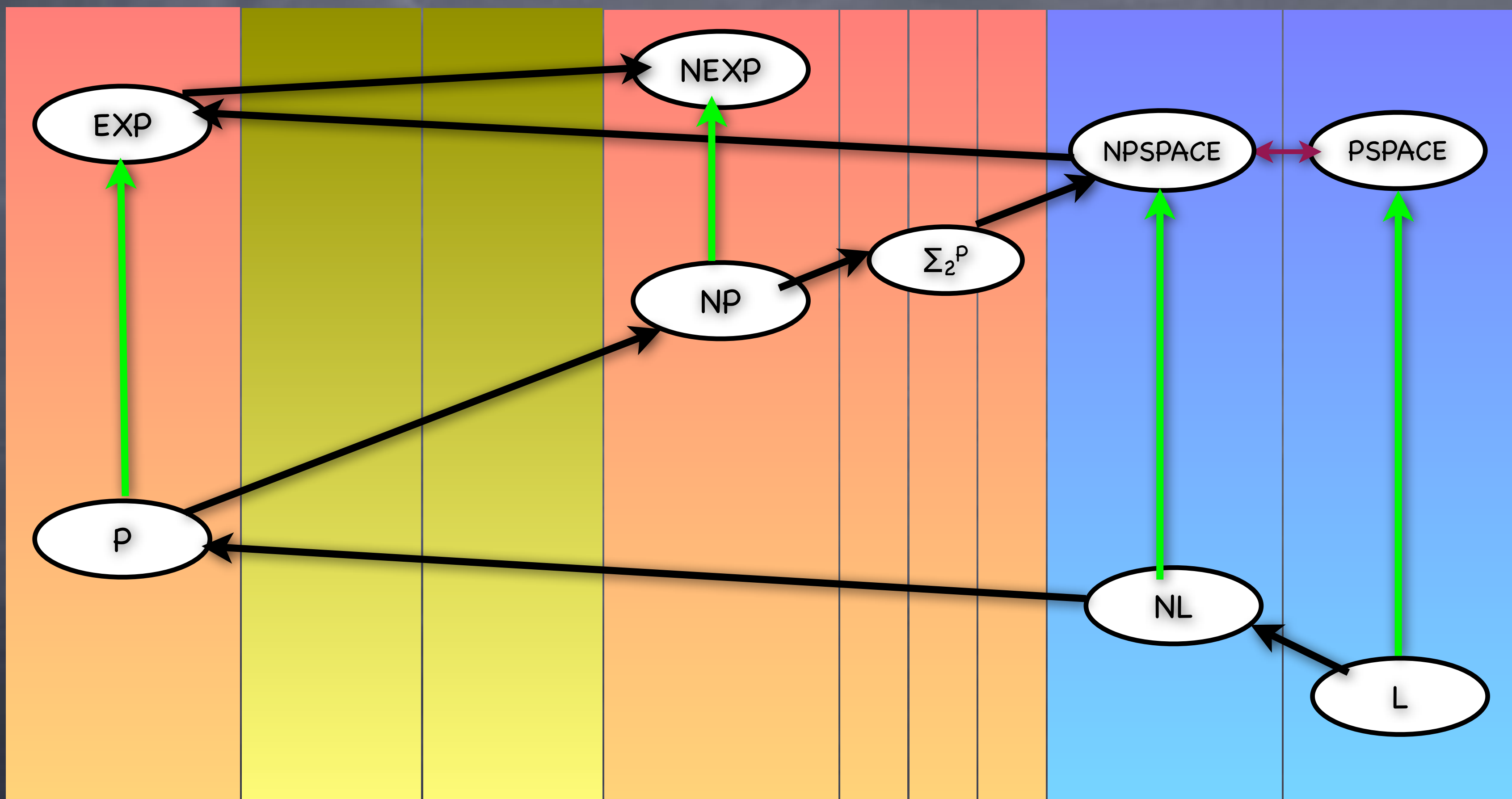
Probabilistic Computation

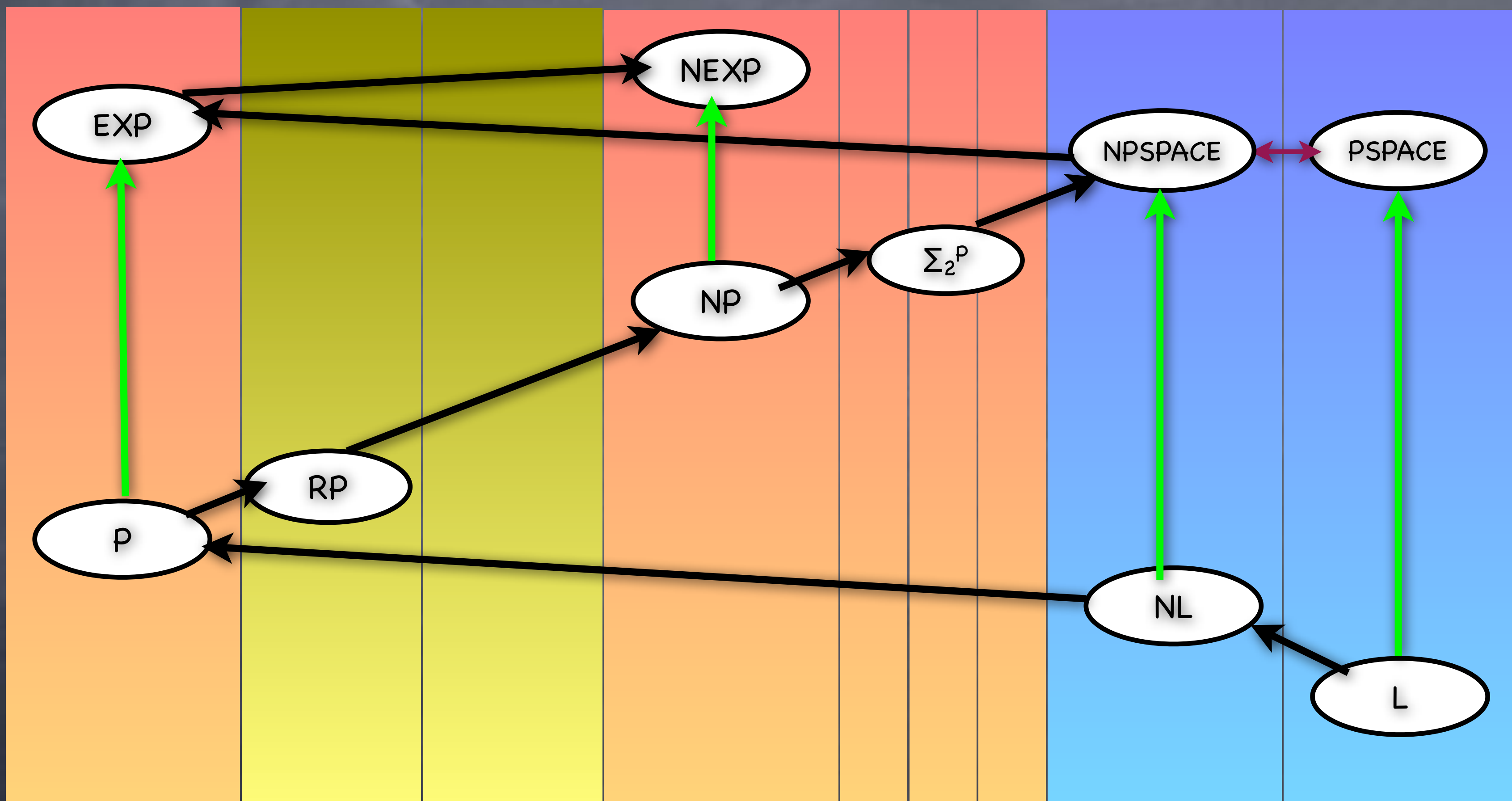
Lecture 14

BPP, ZPP

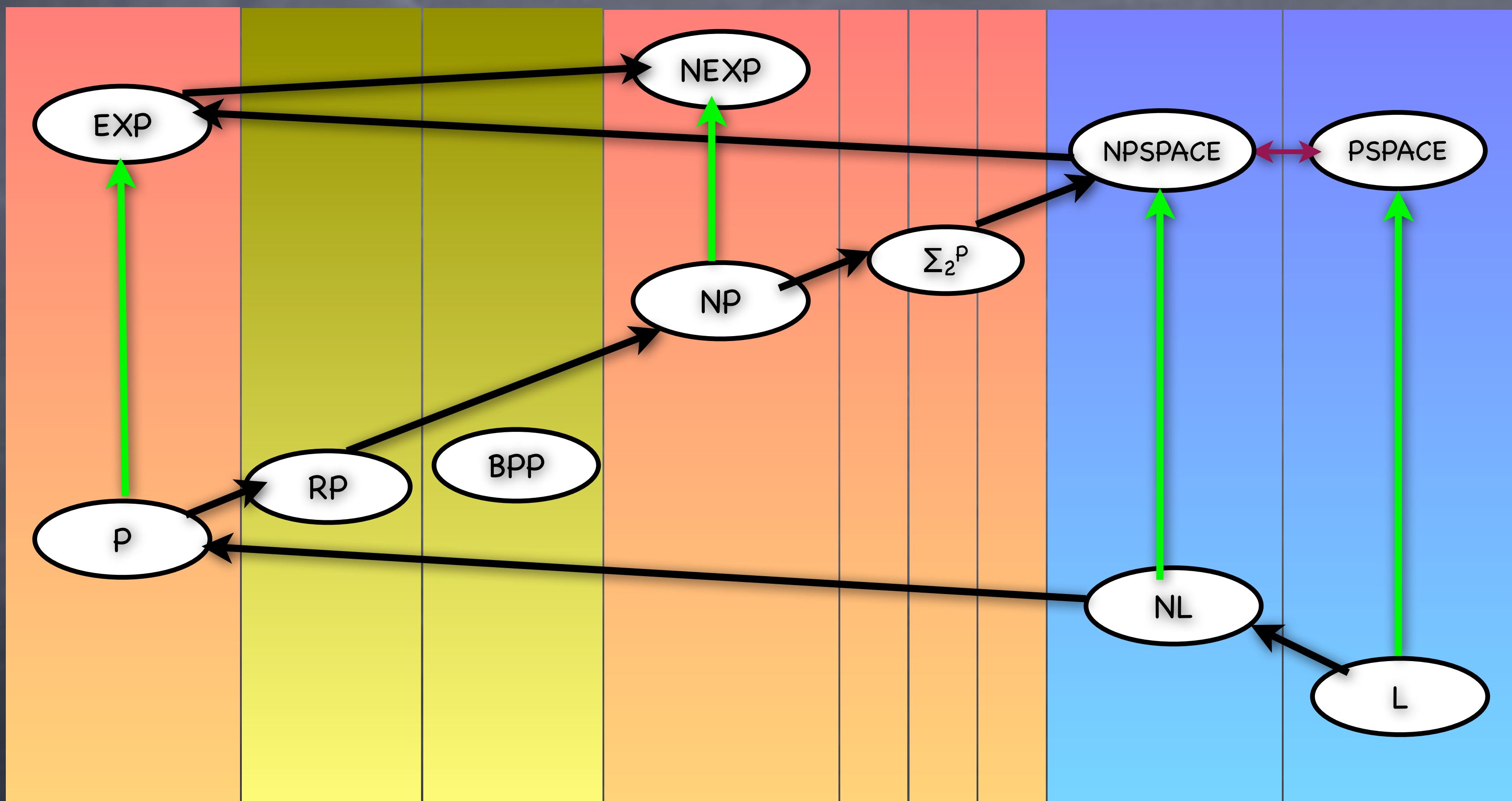
Zoo



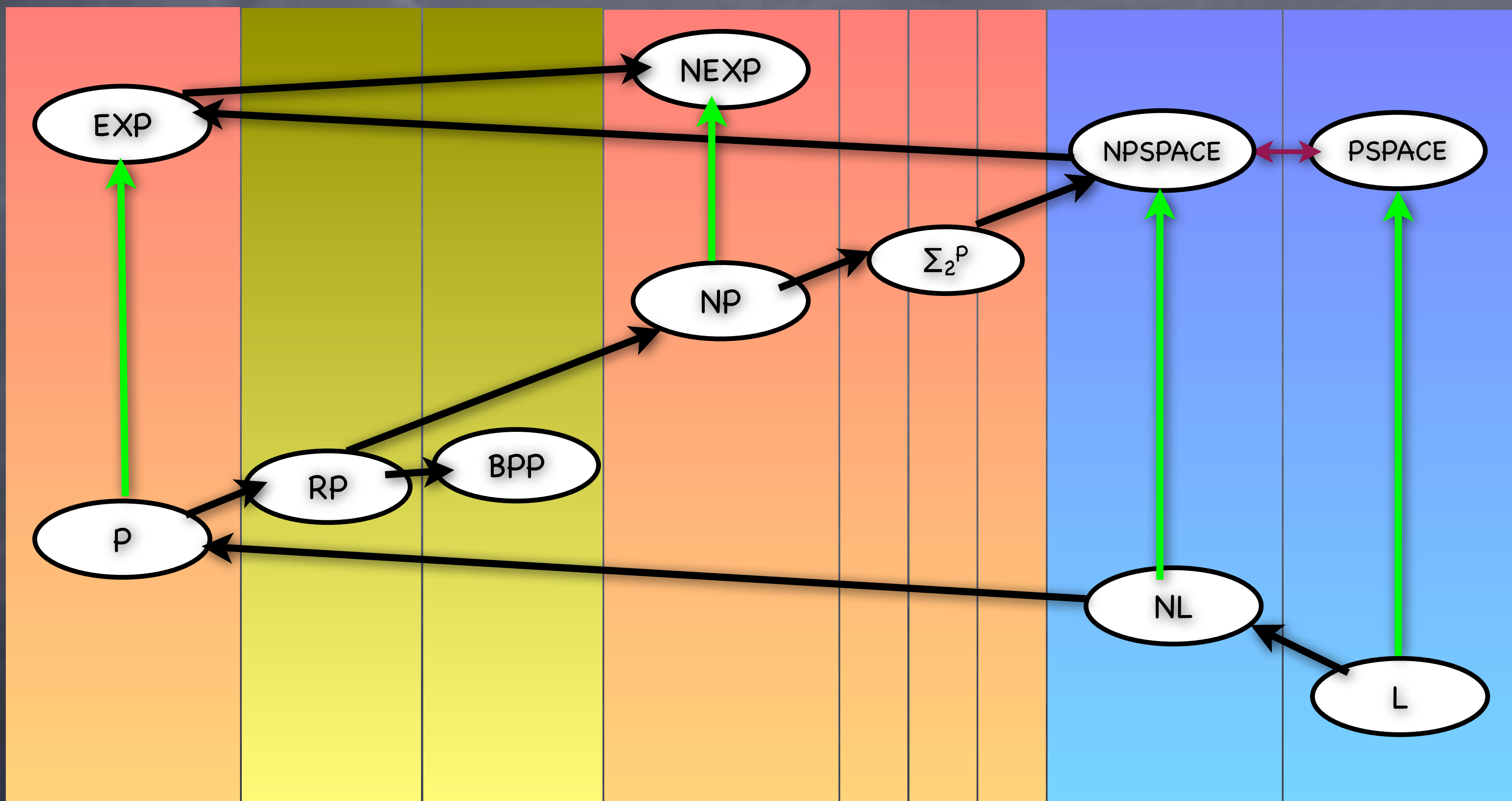
Zoo



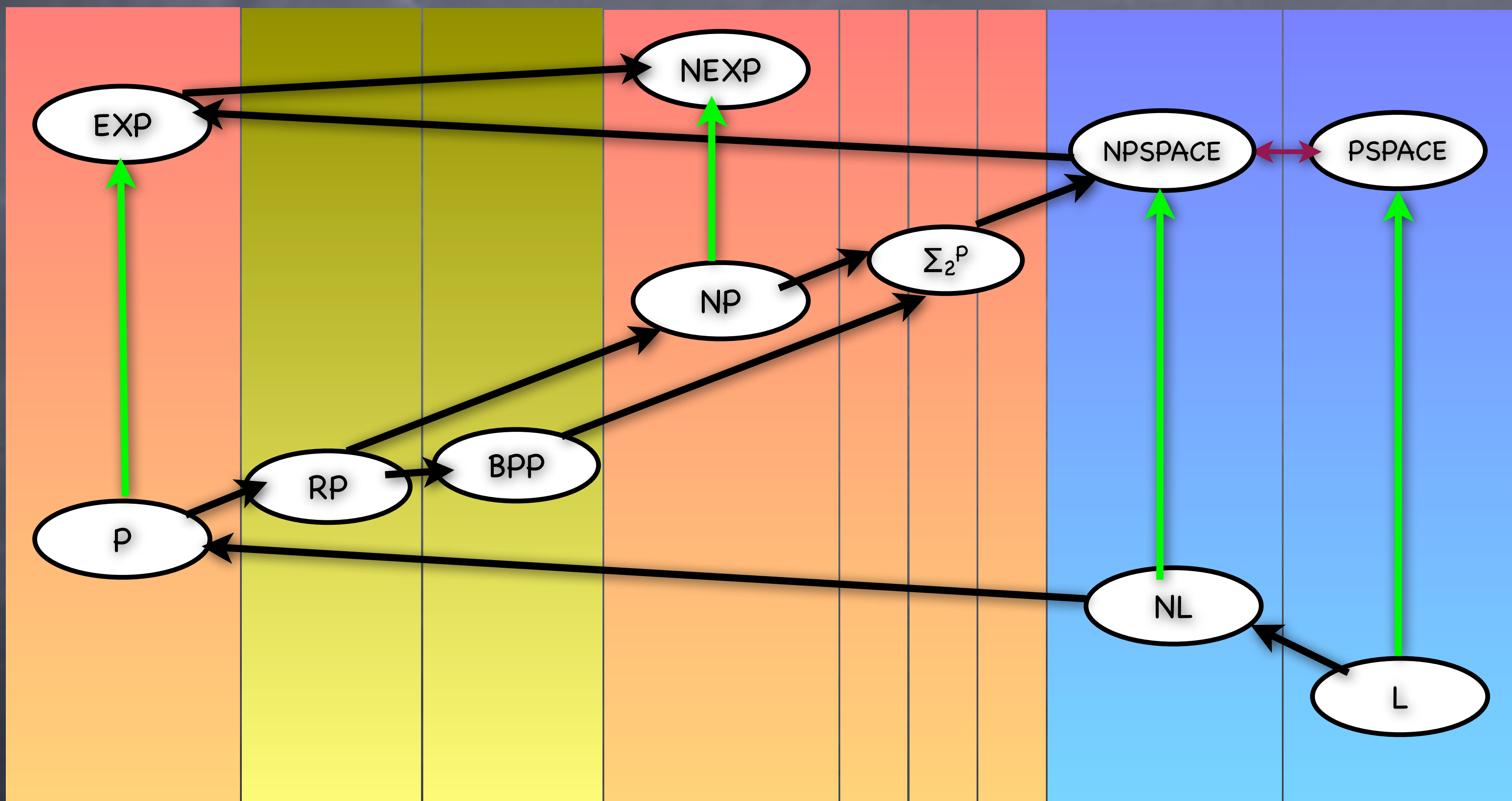
Zoo



Zoo



Zoo



BPP-Complete Problem?

BPP-Complete Problem?

- Not known!

BPP-Complete Problem?

- Not known!
  - $L = \{ (M,x,1^t) \mid M(x)=\text{yes in time } t \text{ with probability} > 2/3\}$?

BPP-Complete Problem?

- Not known!
  - $L = \{ (M,x,1^t) \mid M(x)\text{=yes in time } t \text{ with probability} > 2/3\}$?
  - Is indeed BPP-Hard

BPP-Complete Problem?

- Not known!
  - L = { $(M,x,1^t)$ | $M(x)$=yes in time t with probability > 2/3} ?
  - Is indeed BPP-Hard
  - But in BPP?

BPP-Complete Problem?

- Not known!
  - L = { $(M,x,1^t)$ | M(x)=yes in time t with probability > 2/3} ?
  - Is indeed BPP-Hard
  - But in BPP?
    - Just run M(x) for t steps and accept if it accepts?

BPP-Complete Problem?

- Not known!
  - L = { $(M,x,1^t)$ | M(x)=yes in time t with probability > 2/3} ?
  - Is indeed BPP-Hard
  - But in BPP?
    - Just run M(x) for t steps and accept if it accepts?
      - If $(M,x,1^t)$ in L, we will indeed accept with prob. > 2/3

BPP-Complete Problem?

- Not known!
  - $L = \{ (M,x,1^t) \mid M(x)=\text{yes in time } t \text{ with probability} > 2/3\}$?
  - Is indeed BPP-Hard
  - But in BPP?
    - Just run M(x) for t steps and accept if it accepts?
      - If $(M,x,1^t)$ in L, we will indeed accept with prob. > 2/3
      - But M may not have a bounded gap. Then, if $(M,x,1^t)$ not in L, we may accept with prob. very close to 2/3.

BPTIME-Hierarchy Theorem?

BPTIME-Hierarchy Theorem?

- $\mathrm{BPTIME}(n) \subsetneq \mathrm{BPTIME}(n^{100})$?

BPTIME-Hierarchy Theorem?

- $\mathrm{BPTIME}(n) \subsetneq \mathrm{BPTIME}(n^{100})$?
- Not known!

BPTIME-Hierarchy Theorem?

- BPTIME(n) $\subsetneq$ BPTIME(n^{100})?
- Not known!
  - But is true for BPTIME(T)/1

Some Probabilistic Algorithmic Concepts

Some Probabilistic Algorithmic Concepts

- Sampling to determine some probability

Some Probabilistic Algorithmic Concepts

- Sampling to determine some probability
  - Checking if determinant of a symbolic matrix is zero: Substitute random values for the variables and evaluate using Gaussian elimination in polynomial time

Some Probabilistic Algorithmic Concepts

- Sampling to determine some probability
  - Checking if determinant of a symbolic matrix is zero: Substitute random values for the variables and evaluate using Gaussian elimination in polynomial time
  - Polynomial Identity Testing: polynomial given as an arithmetic circuit. Like above, but values can be too large. So work over a random modulus.

Some Probabilistic Algorithmic Concepts

- Sampling to determine some probability
  - Checking if determinant of a symbolic matrix is zero: Substitute random values for the variables and evaluate using Gaussian elimination in polynomial time
  - Polynomial Identity Testing: polynomial given as an arithmetic circuit. Like above, but values can be too large. So work over a random modulus.
- Random Walks (for sampling)

Some Probabilistic Algorithmic Concepts

- Sampling to determine some probability
  - Checking if determinant of a symbolic matrix is zero: Substitute random values for the variables and evaluate using Gaussian elimination in polynomial time
  - Polynomial Identity Testing: polynomial given as an arithmetic circuit. Like above, but values can be too large. So work over a random modulus.
- Random Walks (for sampling)
  - Monte Carlo algorithms for calculations

Some Probabilistic Algorithmic Concepts

- Sampling to determine some probability
  - Checking if determinant of a symbolic matrix is zero: Substitute random values for the variables and evaluate using Gaussian elimination in polynomial time
  - Polynomial Identity Testing: polynomial given as an arithmetic circuit. Like above, but values can be too large. So work over a random modulus.
- Random Walks (for sampling)
  - Monte Carlo algorithms for calculations
  - Reachability tests

Random Walks

Random Walks

- Which nodes does the walk touch and with what probability?

Random Walks

- Which nodes does the walk touch and with what probability?
  - How do these probabilities vary with number of steps

Random Walks

- Which nodes does the walk touch and with what probability?
  - How do these probabilities vary with number of steps
- Analyzing a random walk

Random Walks

- Which nodes does the walk touch and with what probability?
  - How do these probabilities vary with number of steps
- Analyzing a random walk
  - Probability Vector: p

Random Walks

- Which nodes does the walk touch and with what probability?
  - How do these probabilities vary with number of steps
- Analyzing a random walk
  - Probability Vector: p
  - Transition probability matrix: M

Random Walks

- Which nodes does the walk touch and with what probability?
  - How do these probabilities vary with number of steps
- Analyzing a random walk
  - Probability Vector: p
  - Transition probability matrix: M
  - One step of the walk: $p' = Mp$

Random Walks

- Which nodes does the walk touch and with what probability?
  - How do these probabilities vary with number of steps
- Analyzing a random walk
  - Probability Vector: p
  - Transition probability matrix: M
  - One step of the walk: $p' = Mp$
  - After t steps: $p^{(t)} = M^t p$

Space-Bounded Probabilistic Computation

Space-Bounded Probabilistic Computation

- PL, RL, BPL

Space-Bounded Probabilistic Computation

- PL, RL, BPL
  - Logspace analogues of PP, RP, BPP

Space-Bounded Probabilistic Computation

- PL, RL, BPL
  - Logspace analogues of PP, RP, BPP
- Note: RL $\subseteq$ NL, RL $\subseteq$ BPL

Space-Bounded Probabilistic Computation

- PL, RL, BPL
    - Logspace analogues of PP, RP, BPP
- Note: RL $\subseteq$ NL, RL $\subseteq$ BPL
    - Recall NL $\subseteq$ P (because PATH $\in$ P)

Space-Bounded Probabilistic Computation

- PL, RL, BPL
    - Logspace analogues of PP, RP, BPP
- Note: RL $\subseteq$ NL, RL $\subseteq$ BPL
    - Recall NL $\subseteq$ P (because PATH $\in$ P)
    - So RL $\subseteq$ P

Space-Bounded Probabilistic Computation

- PL, RL, BPL
  - Logspace analogues of PP, RP, BPP
- Note: RL $\subseteq$ NL, RL $\subseteq$ BPL
  - Recall NL $\subseteq$ P (because PATH $\in$ P)
  - So RL $\subseteq$ P
  - In fact BPL $\subseteq$ P

BPL $\subseteq$ P

BPL $\subseteq$ P

- Consider the BPL algorithm, on input x, as a random walk over configurations

BPL $\subseteq$ P

- Consider the BPL algorithm, on input x, as a random walk over configurations
  - Construct the transition matrix M

BPL $\subseteq$ P

- Consider the BPL algorithm, on input x, as a random walk over configurations
    - Construct the transition matrix M
        - Size of graph is poly(n), probability values are 0, 0.5 and 1

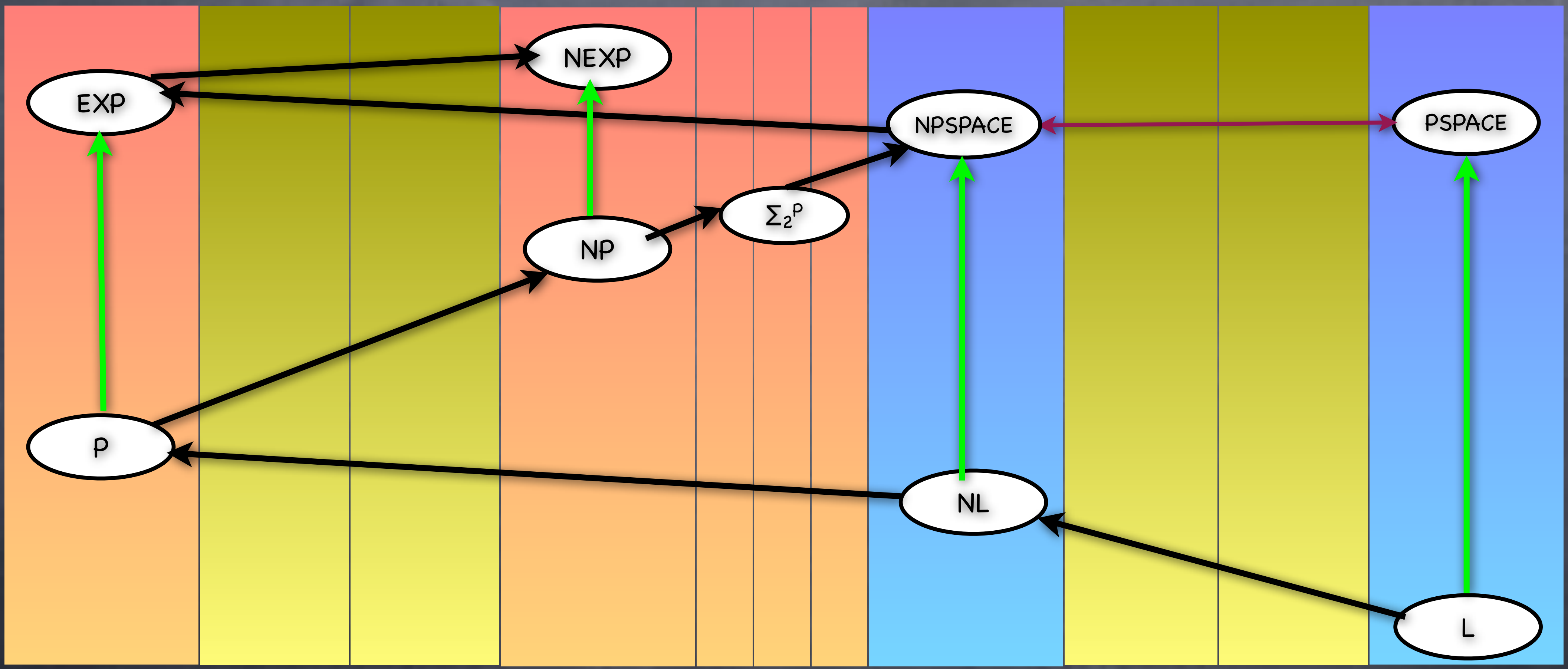
BPL ⊆ P

- Consider the BPL algorithm, on input x, as a random walk over configurations
  - Construct the transition matrix M
    - Size of graph is poly(n), probability values are 0, 0.5 and 1
  - Calculate M^t for t = max running time = poly(n)

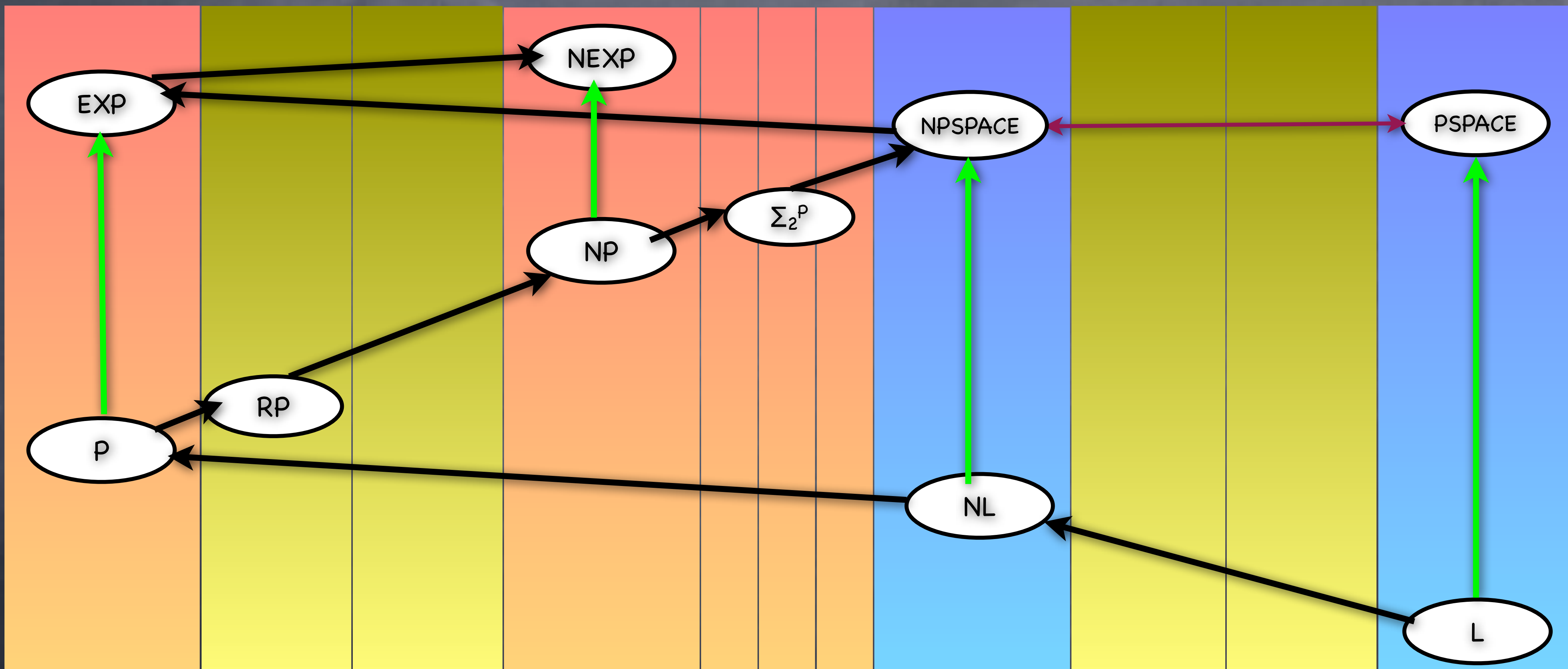
BPL $\subseteq$ P

- Consider the BPL algorithm, on input x, as a random walk over configurations
  - Construct the transition matrix M
    - Size of graph is poly(n), probability values are 0, 0.5 and 1
  - Calculate M^t for t = max running time = poly(n)
  - Accept if $(M^t\, p^{start})_{accept} > 2/3$ where p^{start} is the probability distribution with all the weight on the start configuration

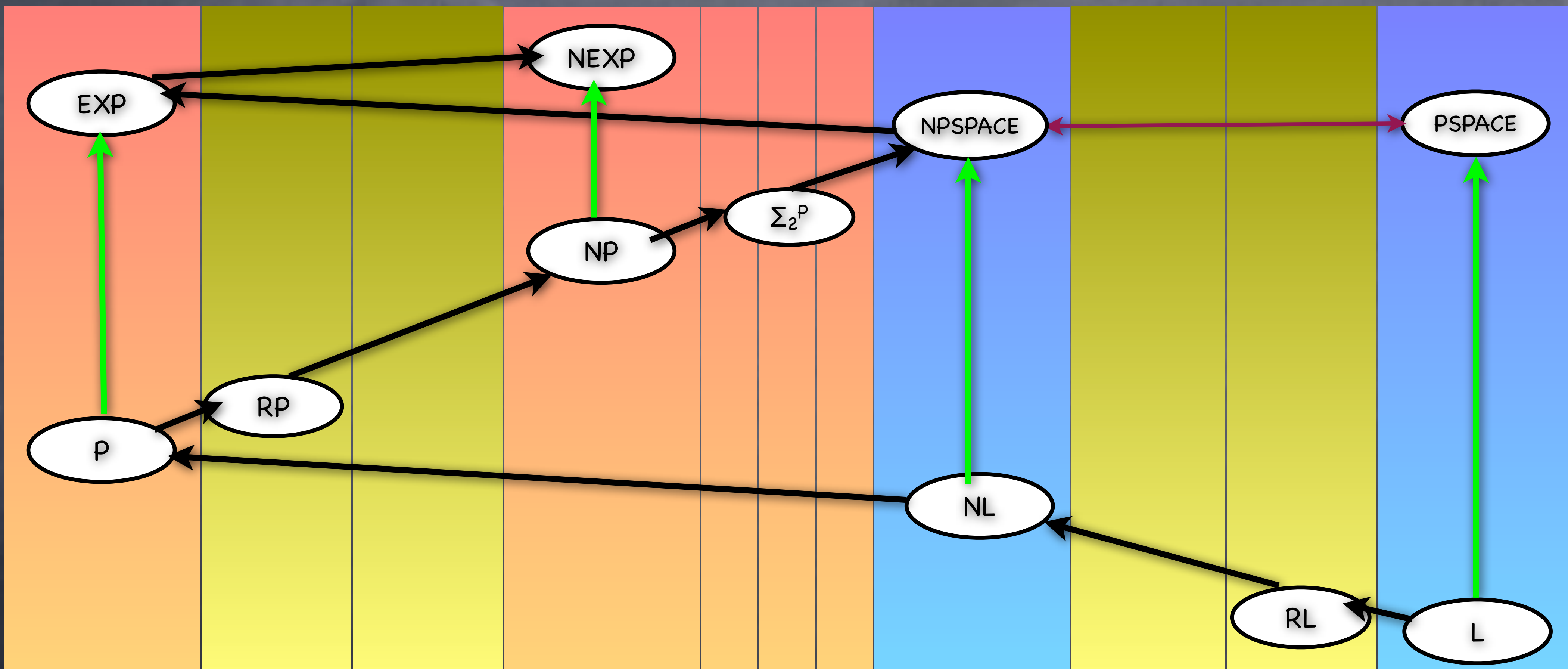
Zoo



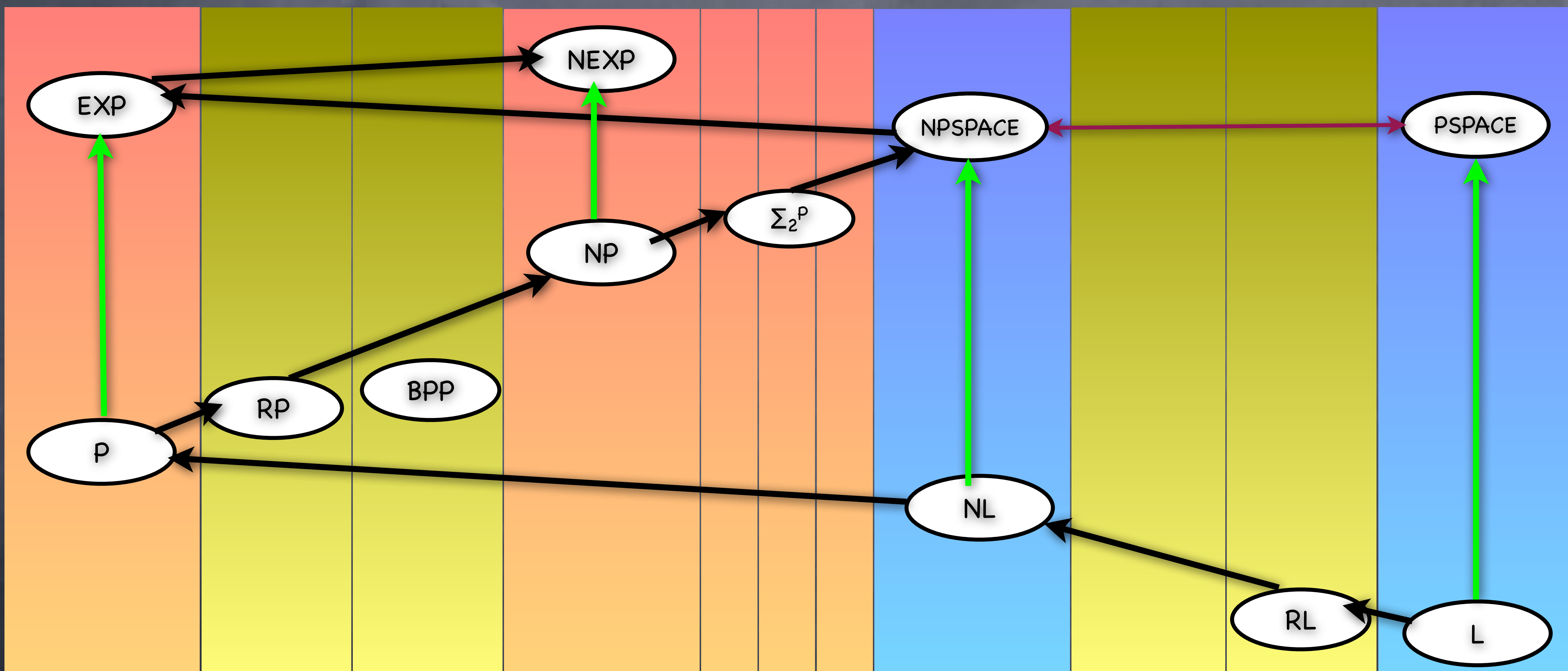
Zoo



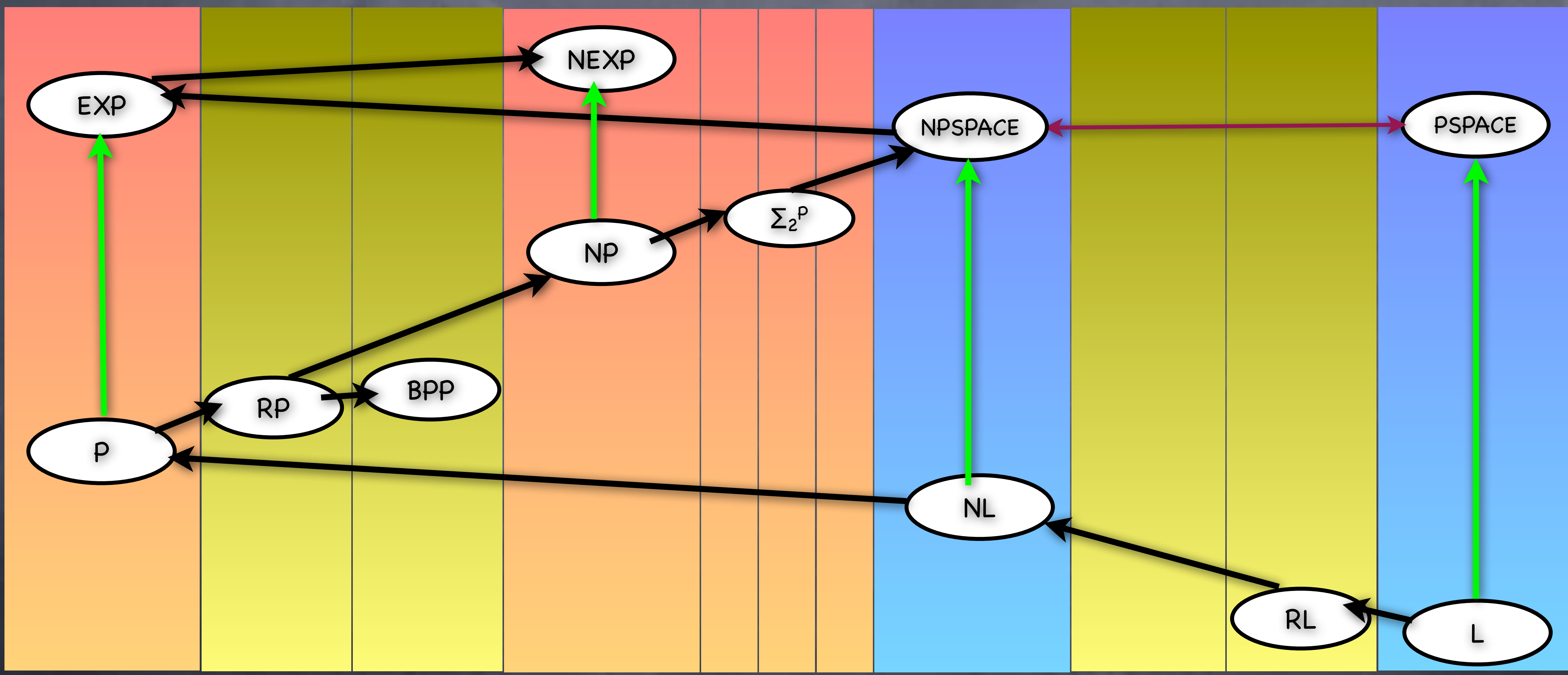
Zoo



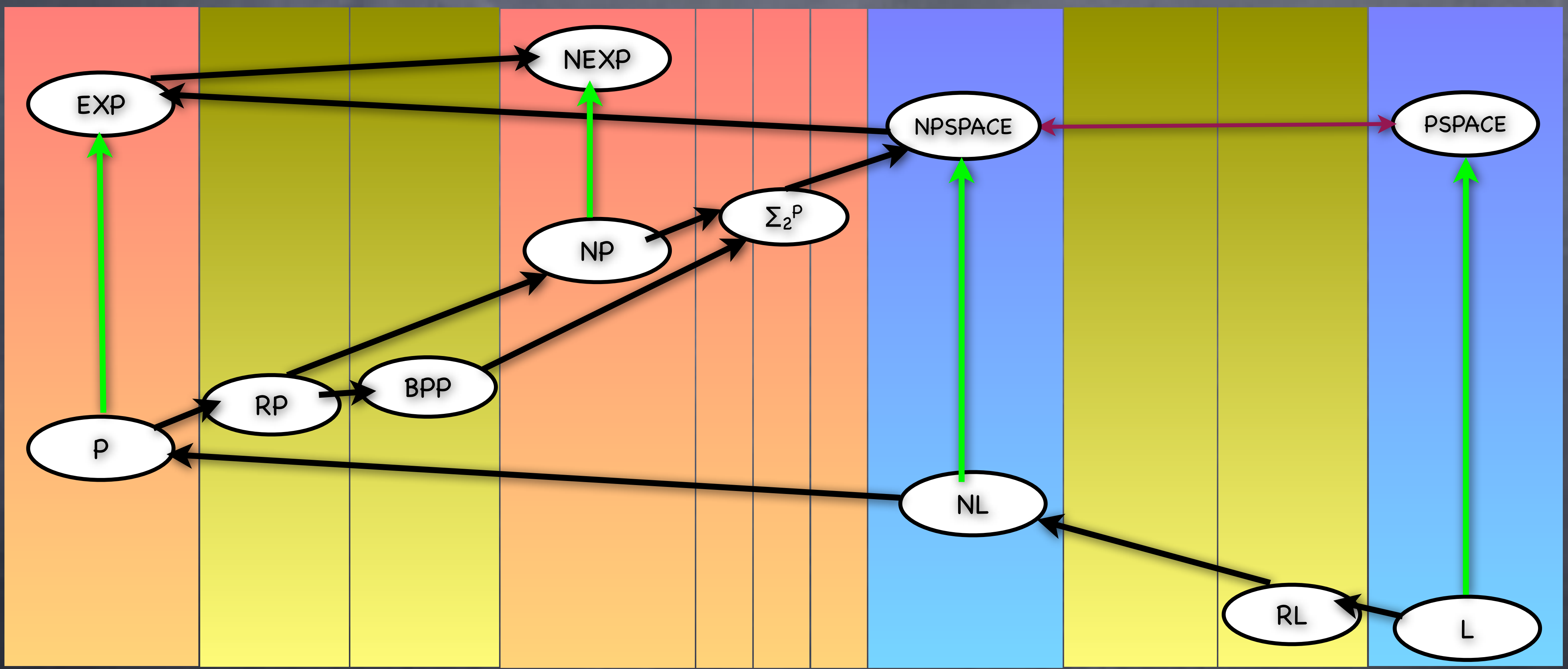
Zoo



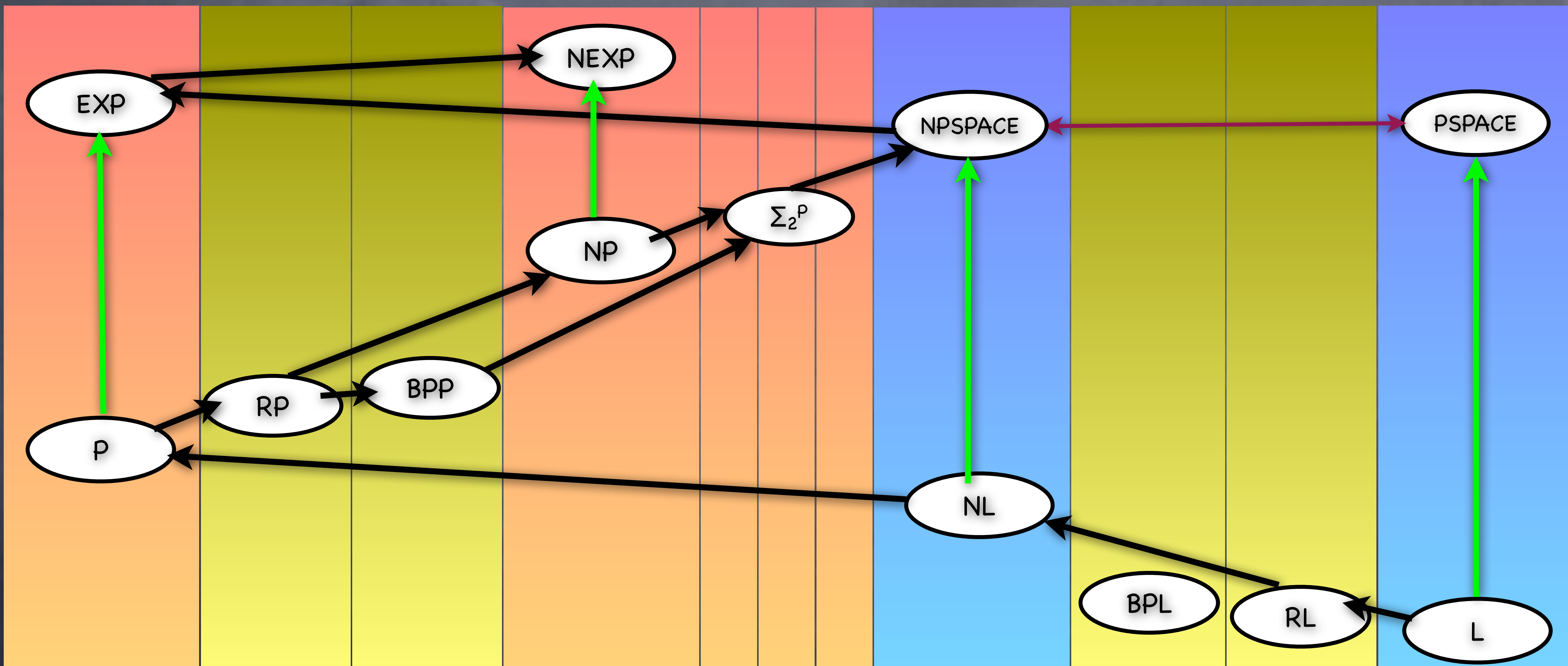
Zoo



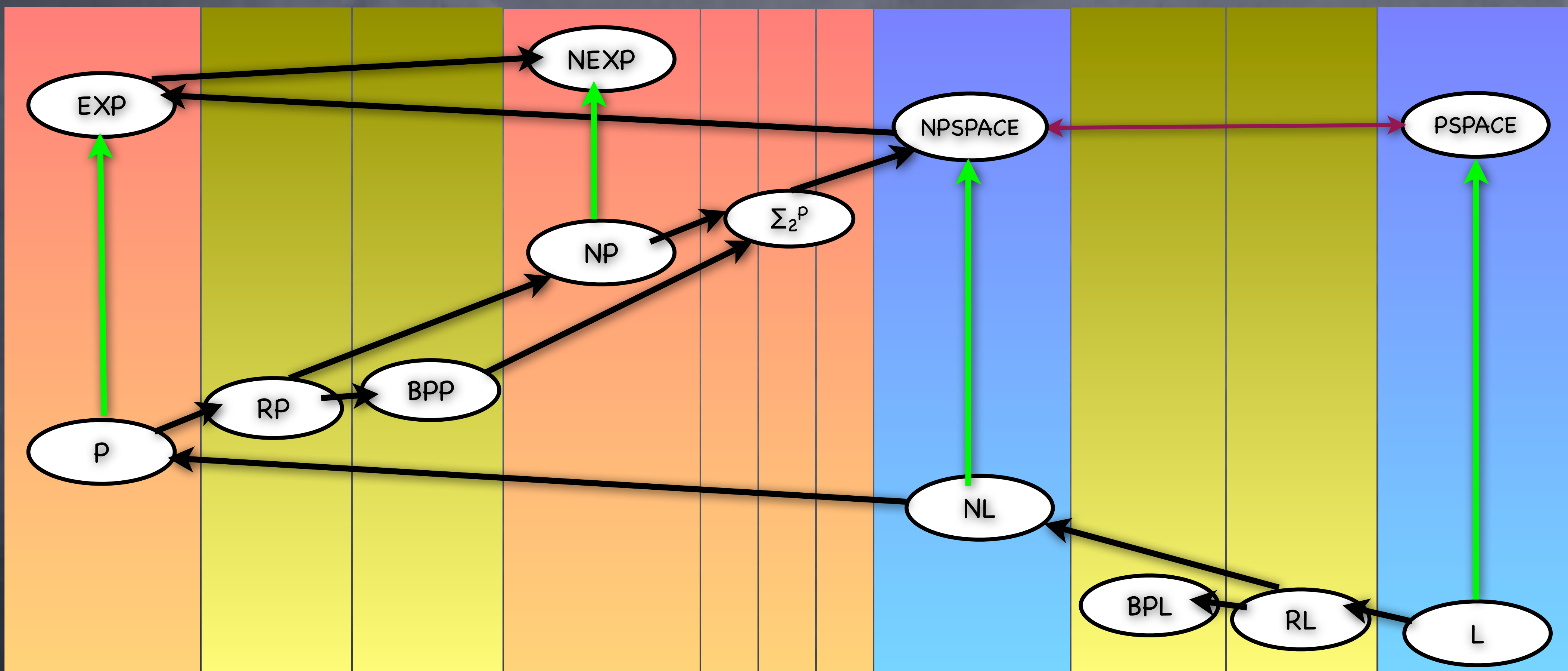
Zoo



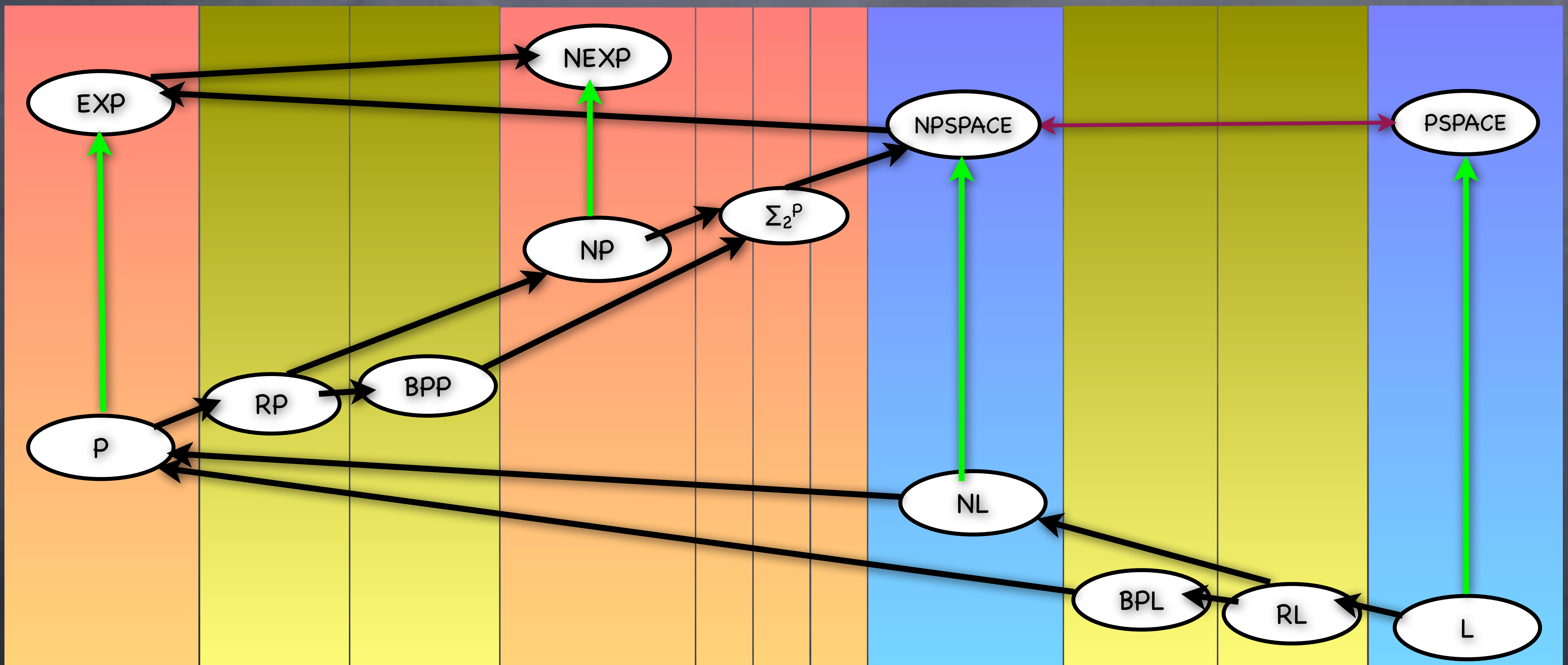
Zoo



Zoo



Zoo



Expected Running Time

Expected Running Time

- Running time is a random variable too

Expected Running Time

- Running time is a random variable too
  - As is the outcome of yes/no

Expected Running Time

- Running time is a random variable too
  - As is the outcome of yes/no
- May ask for running time to be polynomial only in expectation, or with high probability

Expected Running Time

- Running time is a random variable too
    - As is the outcome of yes/no
- May ask for running time to be polynomial only in expectation, or with high probability
- Las Vegas algorithms: only expected running time is polynomial; but when it terminates, it produces the correct answer

Expected Running Time

- Running time is a random variable too
  - As is the outcome of yes/no
- May ask for running time to be polynomial only in expectation, or with high probability
- Las Vegas algorithms: only expected running time is polynomial; but when it terminates, it produces the correct answer
  - Zero error probability

Zero-Error Computation

Zero-Error Computation

- e.g. A simple algorithm for finding median in expected linear time

Zero-Error Computation

- e.g. A simple algorithm for finding median in expected linear time
  - (There are non-trivial algorithms to do it in deterministic linear time. Simple sorting takes O(n log n) time.)

Zero-Error Computation

- e.g. A simple algorithm for finding median in expected linear time
  - (There are non-trivial algorithms to do it in deterministic linear time. Simple sorting takes O(n log n) time.)
- Procedure Find-element(L,k) to find k^{th} smallest element in list L

Zero-Error Computation

- e.g. A simple algorithm for finding median in expected linear time
  - (There are non-trivial algorithms to do it in deterministic linear time. Simple sorting takes O(n log n) time.)
- Procedure Find-element(L,k) to find k^{th} smallest element in list L
  - Pick random element x in L. Scan L; divide it into $L_{>x}$ (elements > x) and $L_{<x}$ (elements < x); also determine position m of x in L.

Zero-Error Computation

- e.g. A simple algorithm for finding median in expected linear time
  - (There are non-trivial algorithms to do it in deterministic linear time. Simple sorting takes $O(n \log n)$ time.)
- Procedure Find-element(L,k) to find k^{th} smallest element in list L
  - Pick random element x in L. Scan L; divide it into $L_{>x}$ (elements $> x$) and $L_{<x}$ (elements $< x$); also determine position m of x in L.
  - If $m = k$, return x. If $m > k$, call Find-element($L_{<x}$,k), else call Find-element($L_{>x}$,k-m)

Zero-Error Computation

- e.g. A simple algorithm for finding median in expected linear time
  - (There are non-trivial algorithms to do it in deterministic linear time. Simple sorting takes O(n log n) time.)
- Procedure Find-element(L,k) to find k^{th} smallest element in list L
  - Pick random element x in L. Scan L; divide it into $L_{>x}$ (elements > x) and $L_{<x}$ (elements < x); also determine position m of x in L.
  - If m = k, return x. If m > k, call Find-element($L_{<x}$,k), else call Find-element($L_{>x}$,k-m)
- Correctness obvious. Expected running time?

Zero-Error Computation

Zero-Error Computation

- Expected running time (worst case over all lists of size n, and all k) be T(n)

Zero-Error Computation

- Expected running time (worst case over all lists of size n, and all k) be T(n)
- Time for non-recursive operations is linear: say bounded by cn. Will show inductively T(n) at most 4cn (base case n=1).

Zero-Error Computation

- Expected running time (worst case over all lists of size n, and all k) be T(n)
- Time for non-recursive operations is linear: say bounded by cn. Will show inductively T(n) at most 4cn (base case n=1).
- $T(n) \leq cn + 1/n \, [\Sigma_{n \geq j > k} T(j) + \Sigma_{0 < j < k} T(n-j)]$

Zero-Error Computation

- Expected running time (worst case over all lists of size n, and all k) be T(n)
- Time for non-recursive operations is linear: say bounded by cn. Will show inductively T(n) at most 4cn (base case n=1).
- $T(n) \leq cn + 1/n\ [\Sigma_{n \geq j > k} T(j) + \Sigma_{0 < j < k} T(n-j)]$
- $T(n) \leq cn + 1/n.4c[\Sigma_{j>k}\ j + \Sigma_{j<k}(n-j)]$ by inductive hypothesis

Zero-Error Computation

- Expected running time (worst case over all lists of size n, and all k) be T(n)
- Time for non-recursive operations is linear: say bounded by cn. Will show inductively T(n) at most 4cn (base case n=1).
- $T(n) \le cn + 1/n\, [\Sigma_{n \ge j > k} T(j) + \Sigma_{0<j<k} T(n-j)]$
- $T(n) \le cn + 1/n.4c[\Sigma_{j>k}\, j + \Sigma_{j<k}(n-j)]$ by inductive hypothesis
- $\Sigma_{j>k}\, j + \Sigma_{j<k}(n-j) = \Sigma_{j>k}\, j + (k-1)n - \Sigma_{j<k}\, j \le \Sigma_j\, j + (k-1)n - 2\, \Sigma_{j<k}\, j$

Zero-Error Computation

- Expected running time (worst case over all lists of size n, and all k) be T(n)
- Time for non-recursive operations is linear: say bounded by cn. Will show inductively T(n) at most 4cn (base case n=1).
- $T(n) \leq cn + 1/n\ [\Sigma_{n \geq j > k} T(j) + \Sigma_{0<j<k} T(n-j)]$
- $T(n) \leq cn + 1/n.4c[\Sigma_{j>k}\ j + \Sigma_{j<k}(n-j)]$ by inductive hypothesis
- $\Sigma_{j>k}\ j + \Sigma_{j<k}(n-j) = \Sigma_{j>k}\ j + (k-1)n - \Sigma_{j<k}\ j \leq \Sigma_j\ j + (k-1)n - 2\ \Sigma_{j<k}\ j$
  - $\leq n^2/2 + (k-1)n - k(k-1) < n^2/2 + k(n-k) \leq 3/4\ n^2$

Zero-Error Computation

- Expected running time (worst case over all lists of size n, and all k) be T(n)
- Time for non-recursive operations is linear: say bounded by cn. Will show inductively T(n) at most 4cn (base case n=1).
- $T(n) \leq cn + 1/n\ [\Sigma_{n \geq j > k} T(j) + \Sigma_{0<j<k} T(n-j)]$
- $T(n) \leq cn + 1/n.4c[\Sigma_{j>k}\ j + \Sigma_{j<k}(n-j)]$ by inductive hypothesis
- $\Sigma_{j>k}\ j + \Sigma_{j<k}(n-j) = \Sigma_{j>k}\ j + (k-1)n - \Sigma_{j<k}\ j \leq \Sigma_j\ j + (k-1)n - 2\ \Sigma_{j<k}\ j$
  - $\leq n^2/2 + (k-1)n - k(k-1) < n^2/2 + k(n-k) \leq 3/4\ n^2$
  - $T(n) \leq cn + 3cn$ as required

Zero-Error Computation

Zero-Error Computation

- Las-Vegas Algorithms: Probabilistic algorithms with deterministic outcome (but probabilistic run time)

Zero-Error Computation

- Las-Vegas Algorithms: Probabilistic algorithms with deterministic outcome (but probabilistic run time)
- ZPTIME(T): class of languages decided by a zero-error probabilistic TM, with expected running time at most T

Zero-Error Computation

- Las-Vegas Algorithms: Probabilistic algorithms with deterministic outcome (but probabilistic run time)
- ZPTIME(T): class of languages decided by a zero-error probabilistic TM, with expected running time at most T
- ZPP = ZPTIME(poly)

Zero-Error Computation

- Las-Vegas Algorithms: Probabilistic algorithms with deterministic outcome (but probabilistic run time)
- ZPTIME(T): class of languages decided by a zero-error probabilistic TM, with expected running time at most T
- ZPP = ZPTIME(poly)
  - ZPP = RP ∩ co-RP

ZPP $\subseteq$ RP

ZPP $\subseteq$ RP

- Truncate after "long enough," and say "no"

ZPP $\subseteq$ RP

- Truncate after "long enough," and say "no"
- Do we still have bounded (one-sided) error?

ZPP $\subseteq$ RP

- Truncate after "long enough," and say "no"
- Do we still have bounded (one-sided) error?
- Will run for "too long" only with small probability

ZPP $\subseteq$ RP

- Truncate after "long enough," and say "no"
- Do we still have bounded (one-sided) error?
- Will run for "too long" only with small probability
  - Because expected running time short

ZPP $\subseteq$ RP

- Truncate after "long enough," and say "no"
- Do we still have bounded (one-sided) error?
- Will run for "too long" only with small probability
  - Because expected running time short
  - With high probability the running time does not exceed the expected running time by much

ZPP $\subseteq$ RP

- Truncate after "long enough," and say "no"
- Do we still have bounded (one-sided) error?
- Will run for "too long" only with small probability
  - Because expected running time short
  - With high probability the running time does not exceed the expected running time by much
  - Pr[X > a E[X]] < 1/a   (non-negative X)

ZPP $\subseteq$ RP

- Truncate after "long enough," and say "no"
- Do we still have bounded (one-sided) error?
- Will run for "too long" only with small probability
  - Because expected running time short
  - With high probability the running time does not exceed the expected running time by much
  - $\Pr[\, X > a\, E[X]\,] < 1/a$ (non-negative X)
  - Markov's inequality

ZPP ⊆ RP

- Truncate after "long enough," and say "no"
- Do we still have bounded (one-sided) error?
- Will run for "too long" only with small probability
    - Because expected running time short
    - With high probability the running time does not exceed the expected running time by much
    - Pr[X > a E[X]] < 1/a (non-negative X)
    - Markov's inequality
- Pr[error] at most 1/a if truncated after a times expected running time

$RP \cap co\text{-}RP \subseteq ZPP$

RP $\cap$ co-RP $\subseteq$ ZPP

- If $L \in$ RP $\cap$ co-RP, then a ZPP algorithm for L:
  - Run both RP and coRP algorithms
  - If former says yes or latter says no, output that answer
  - Else, i.e., if former says no and latter yes, repeat
    - Expected number of repeats = O(1)

Today

Today

- Zoo
  - BPL $\subseteq$ P
- Expected running time
- Zero-Error probabilistic computation
- ZPP = RP $\cap$ co-RP