Computational Complexity

Lecture 9

More of the Polynomial Hierarchy

Alternation

PH is in terms of verification

PH is in terms of verification

- Recall Σ_k^P

PH is in terms of verification

- Recall Σ_k^P
  - Languages $L = \{x| \exists w_1 \forall w_2 \ldots Q w_k\ F(x;w_1,w_2,..,w_k)\}$, where F in P

PH is in terms of verification

- Recall Σ_k^P
  - Languages $L = \{x | \exists w_1 \forall w_2 ... Q w_k\ F(x; w_1, w_2, .., w_k)\}$, where F in P
  - Consider deterministic polynomial time machine M for F, with k read-once tapes for the certificates

PH is in terms of verification

- Recall Σ_k^p
  - Languages $L = \{x \mid \exists w_1 \forall w_2 \ldots Q w_k\ F(x; w_1, w_2, .., w_k)\}$, where F in P
  - Consider deterministic polynomial time machine M for F, with k read-once tapes for the certificates
    - Tapes read one after the other

PH is in terms of verification

- Recall Σ_k^P
  - Languages $L = \{x| \exists w_1 \forall w_2 ... Qw_k \; F(x;w_1,w_2,..,w_k)\}$, where F in P
  - Consider deterministic polynomial time machine M for F, with k read-once tapes for the certificates
    - Tapes read one after the other
  - x in L if $\exists w_1 \; \forall w_2 \; ... \; Qw_k$ such that $M(x;w_1,w_2,..,w_k)$ accepts

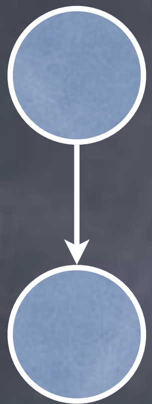
PH is in terms of verification

- Recall Σ_k^P
  - Languages $L = \{x | \exists w_1 \forall w_2 ... Q w_k\ F(x; w_1, w_2, .., w_k)\}$, where F in P
  - Consider deterministic polynomial time machine M for F, with k read-once tapes for the certificates
    - Tapes read one after the other
  - x in L if $\exists w_1\ \forall w_2\ ...\ Q w_k$ such that $M(x; w_1, w_2, .., w_k)$ accepts
- Plan: Formulate in terms of a non-deterministic TM (with no certificates)

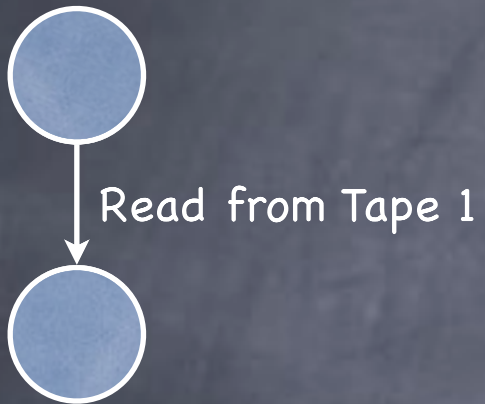
Verification → Non-determinism

Verification →
Non-determinism

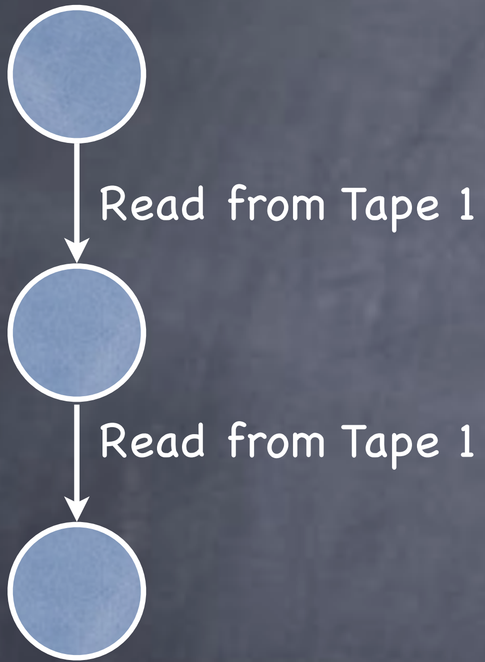
Verification → Non-determinism



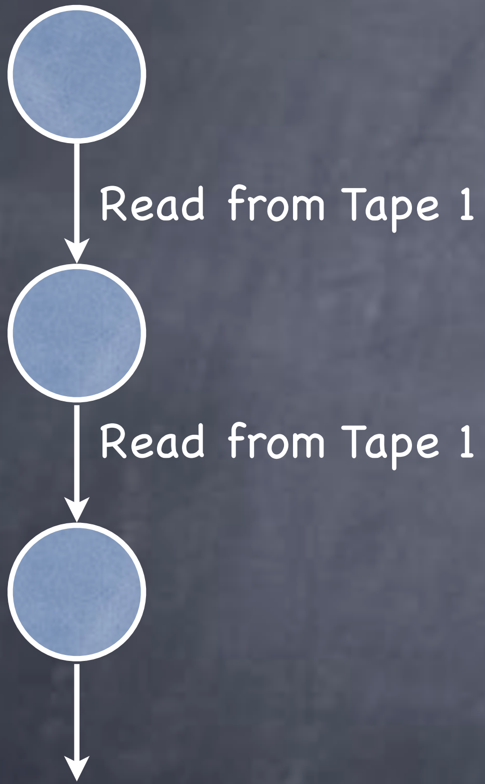
Verification → Non-determinism



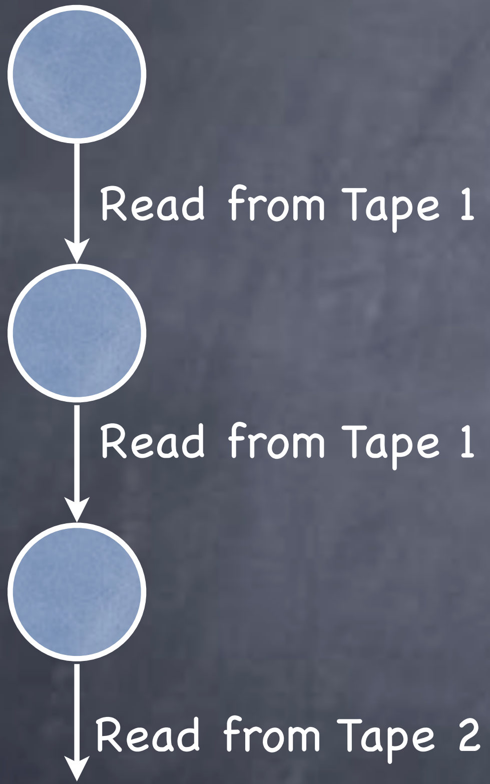
Verification → Non-determinism



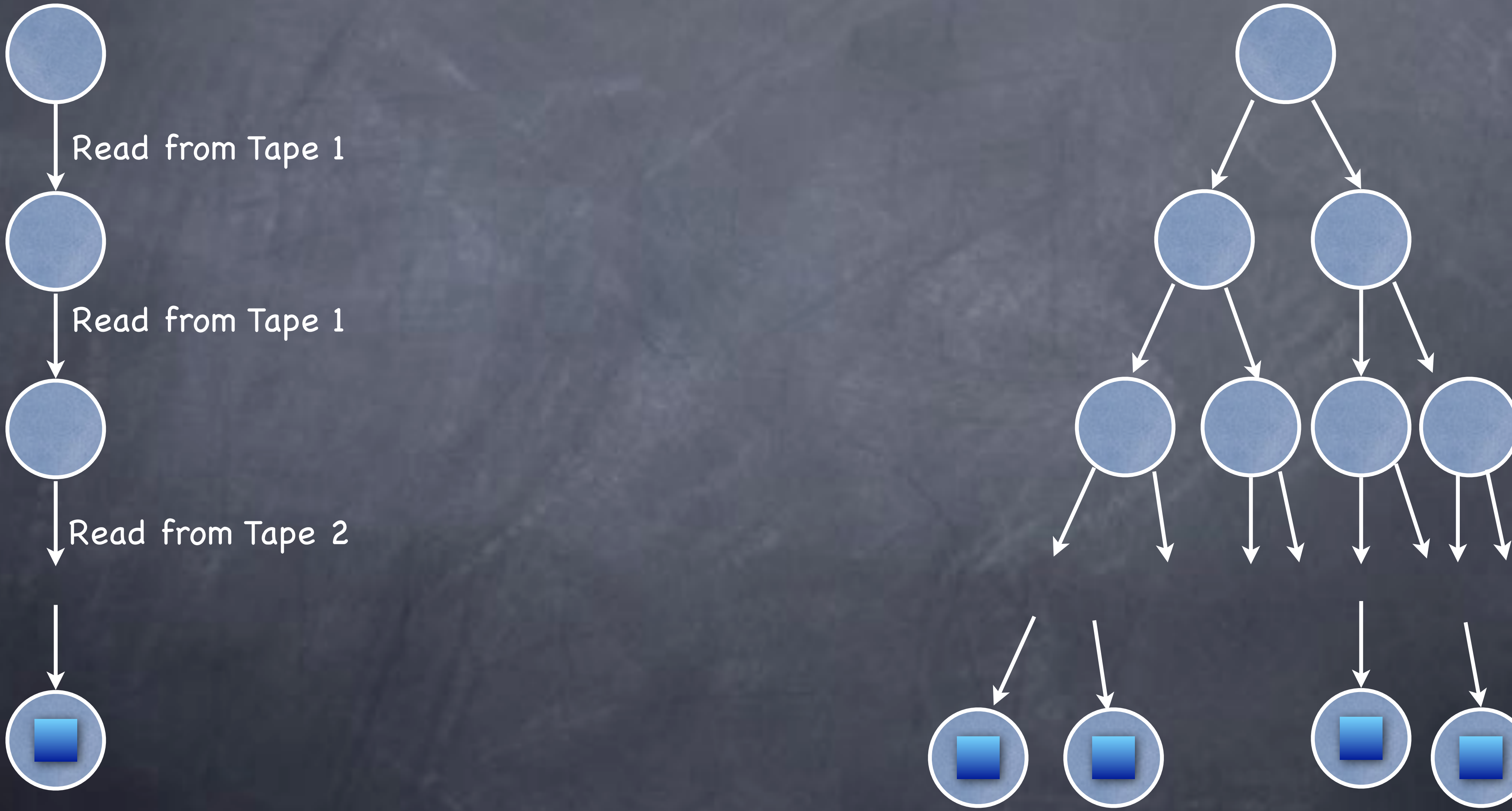
Verification → Non-determinism



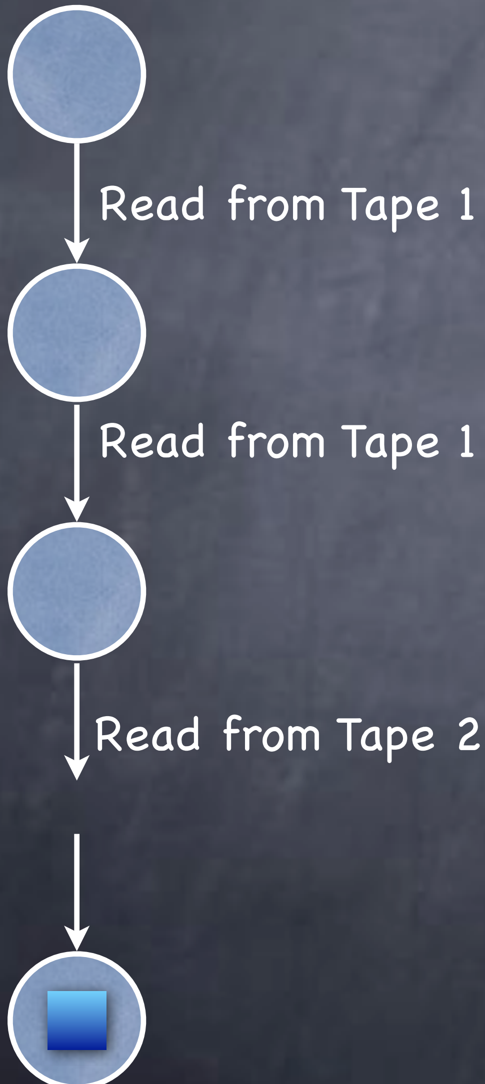
Verification → Non-determinism

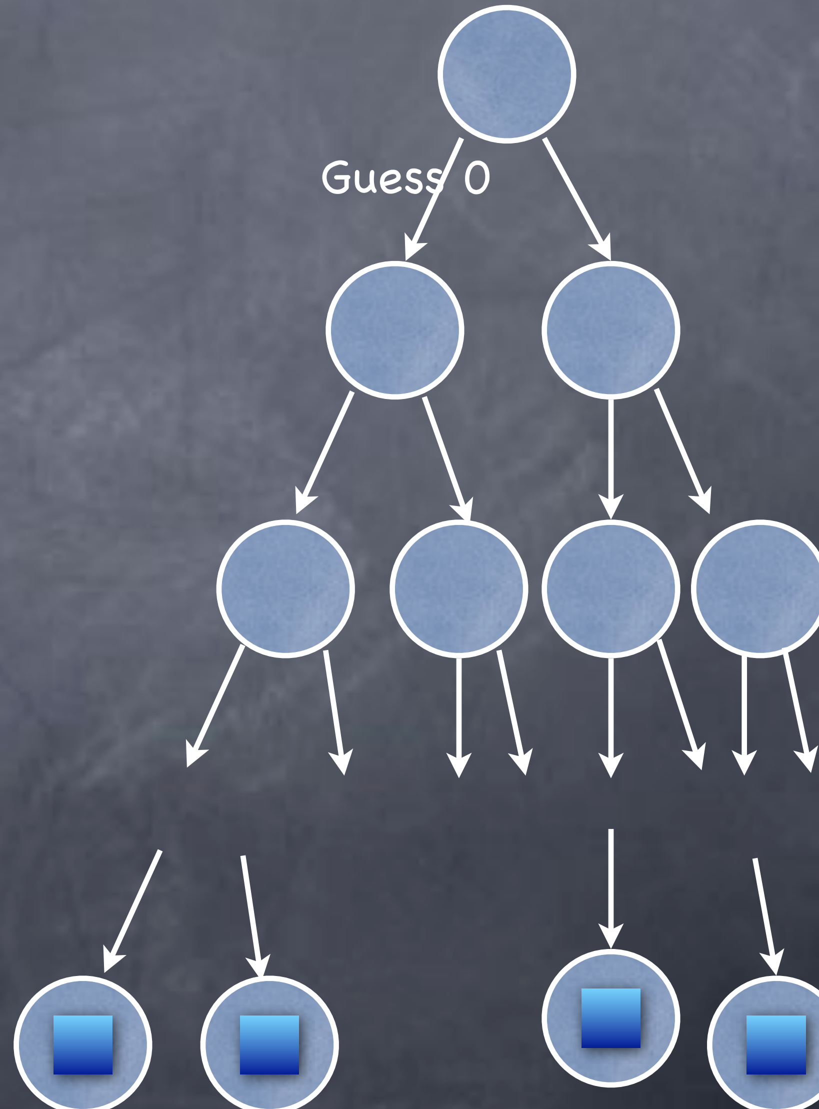


Verification → Non-determinism

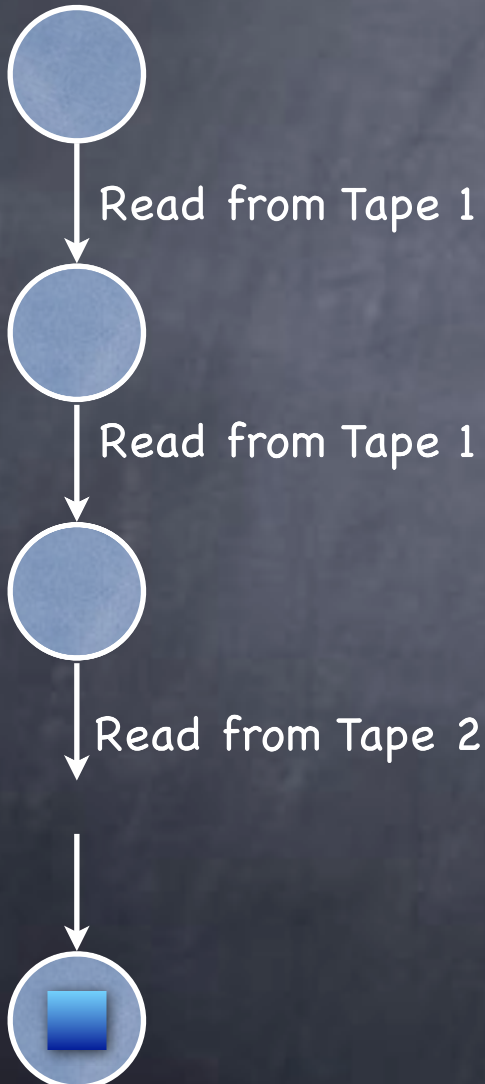


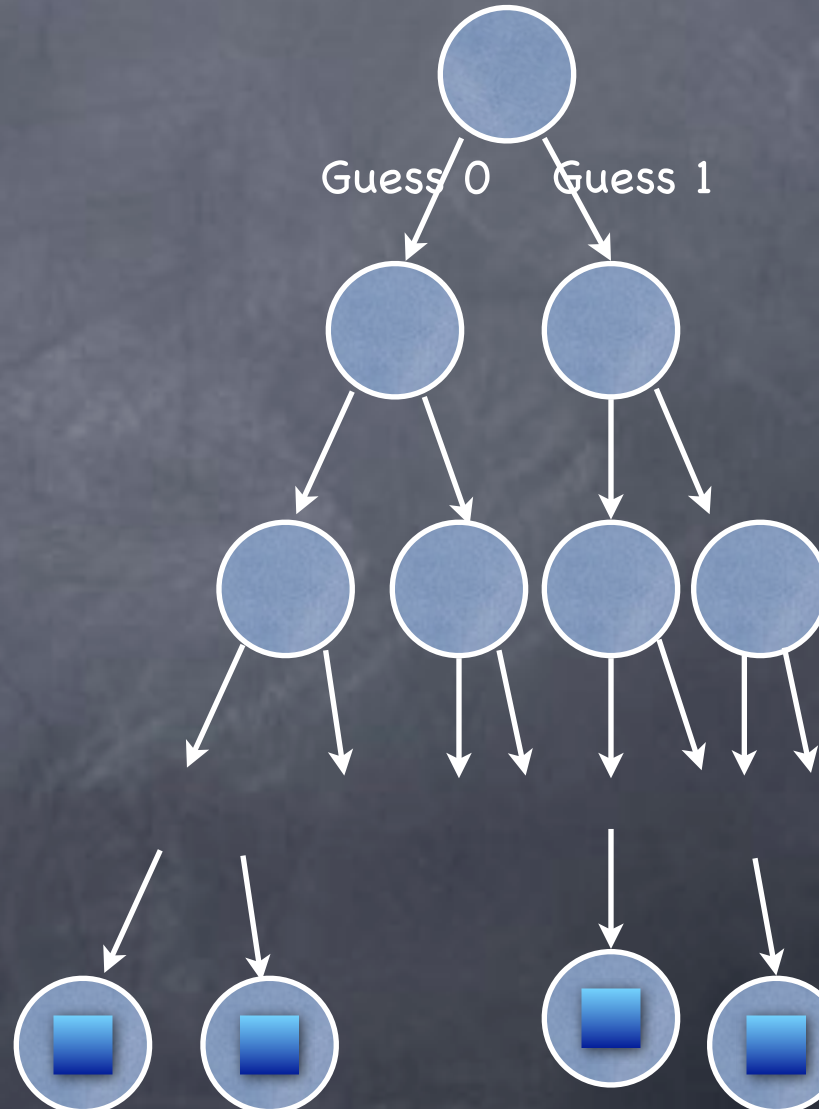
Verification → Non-determinism



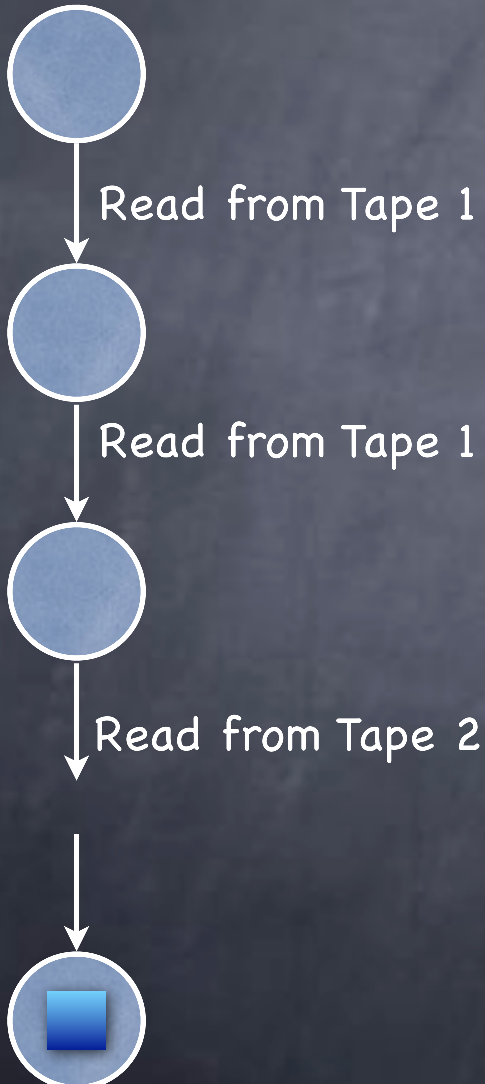


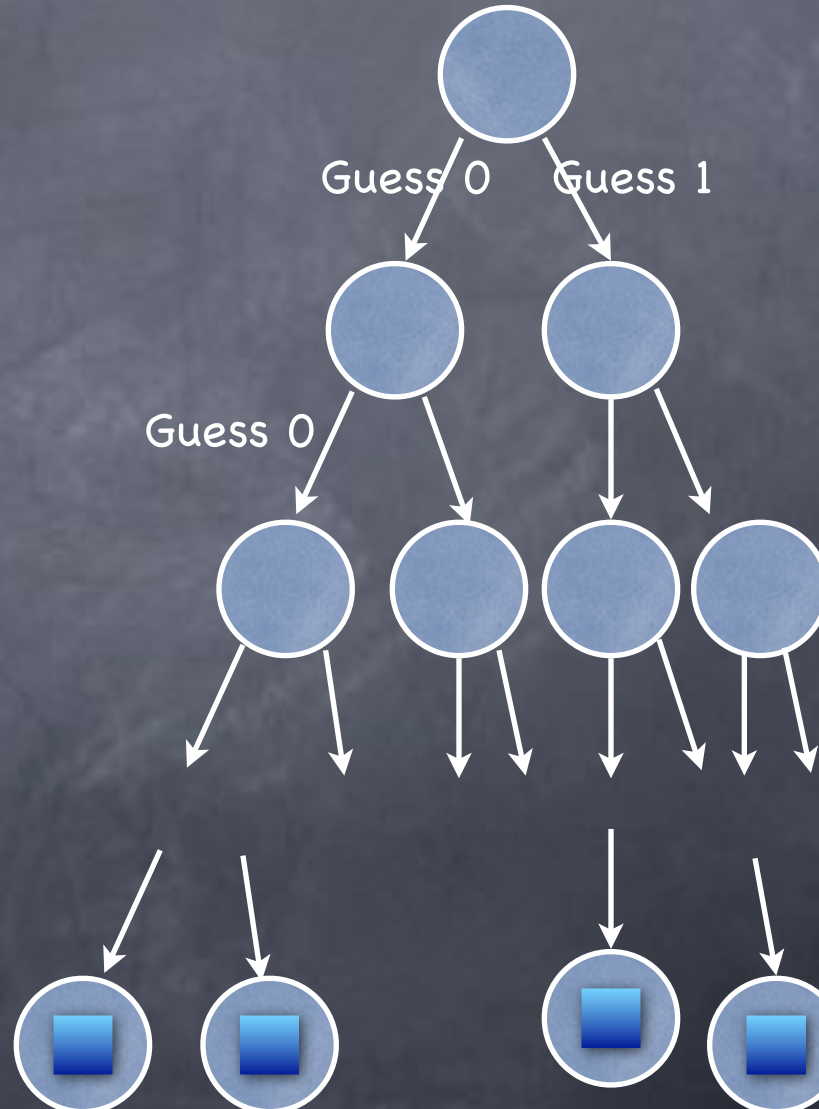
Verification → Non-determinism



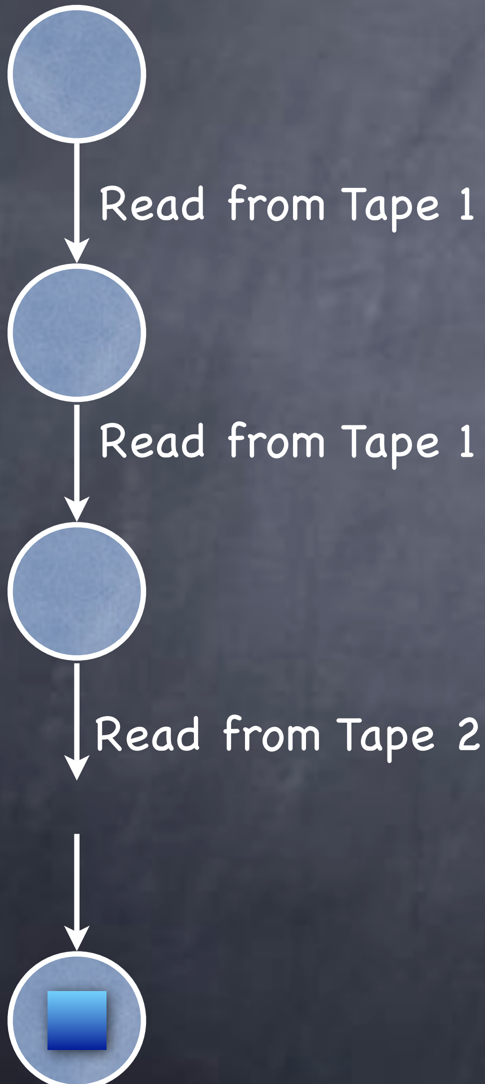


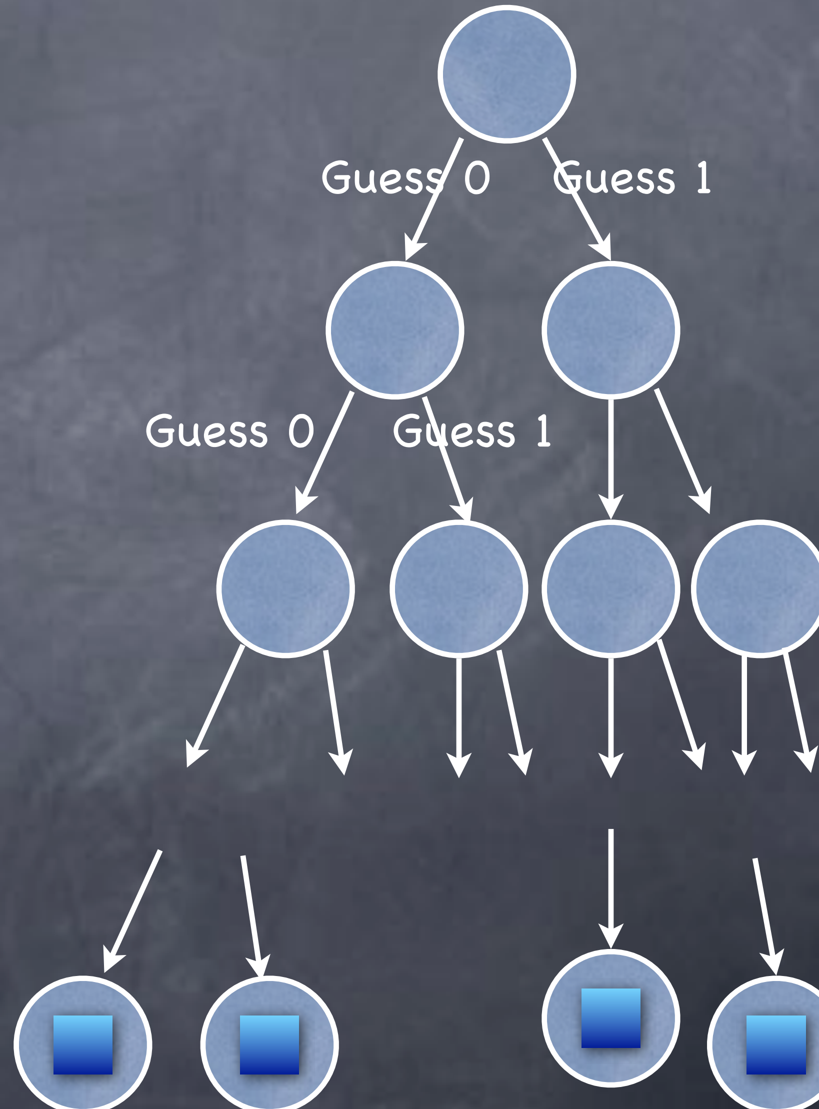
Verification → Non-determinism



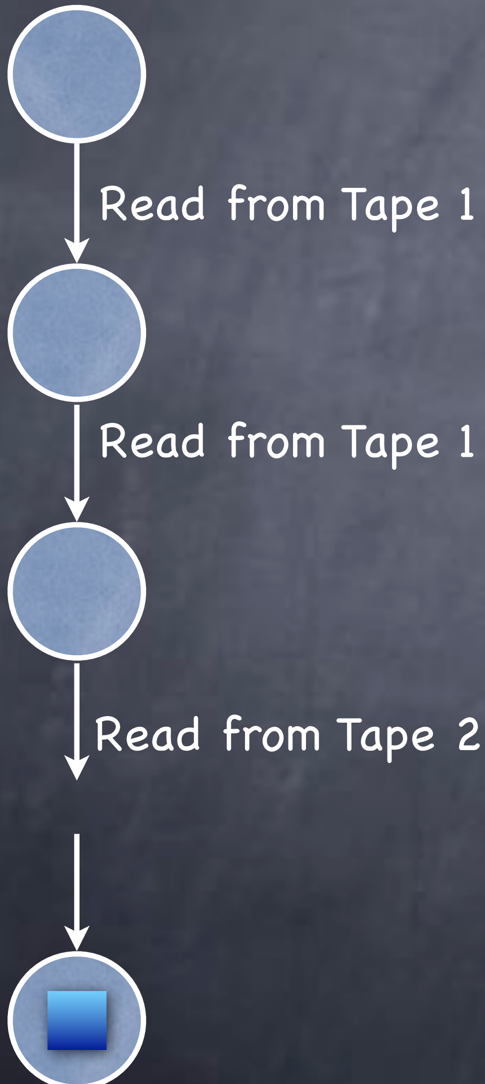


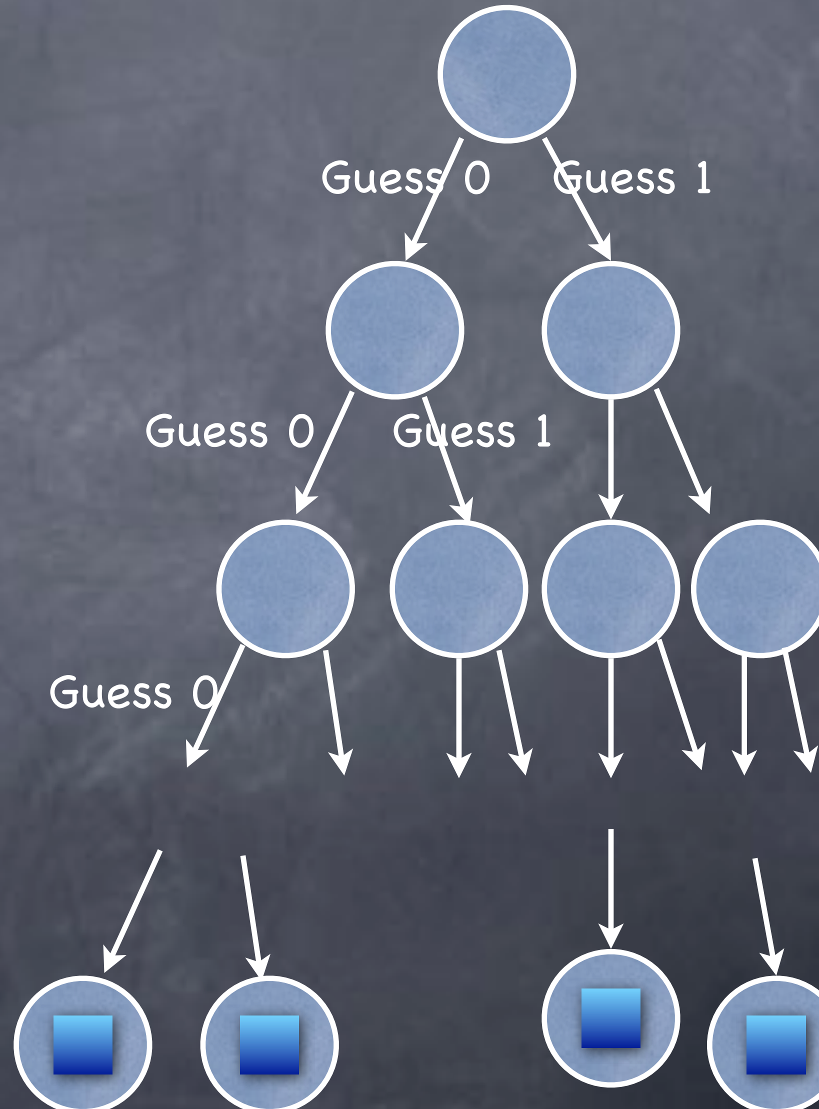
Verification → Non-determinism



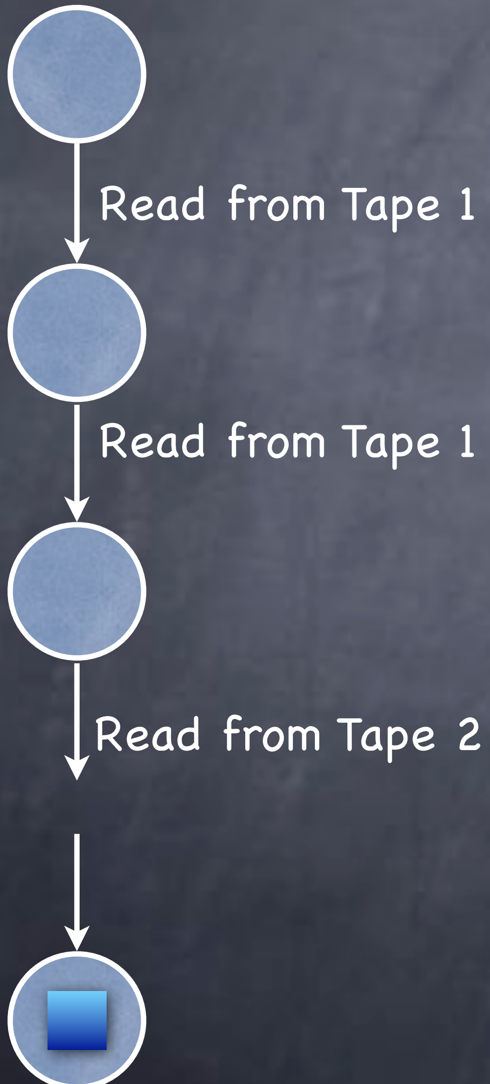


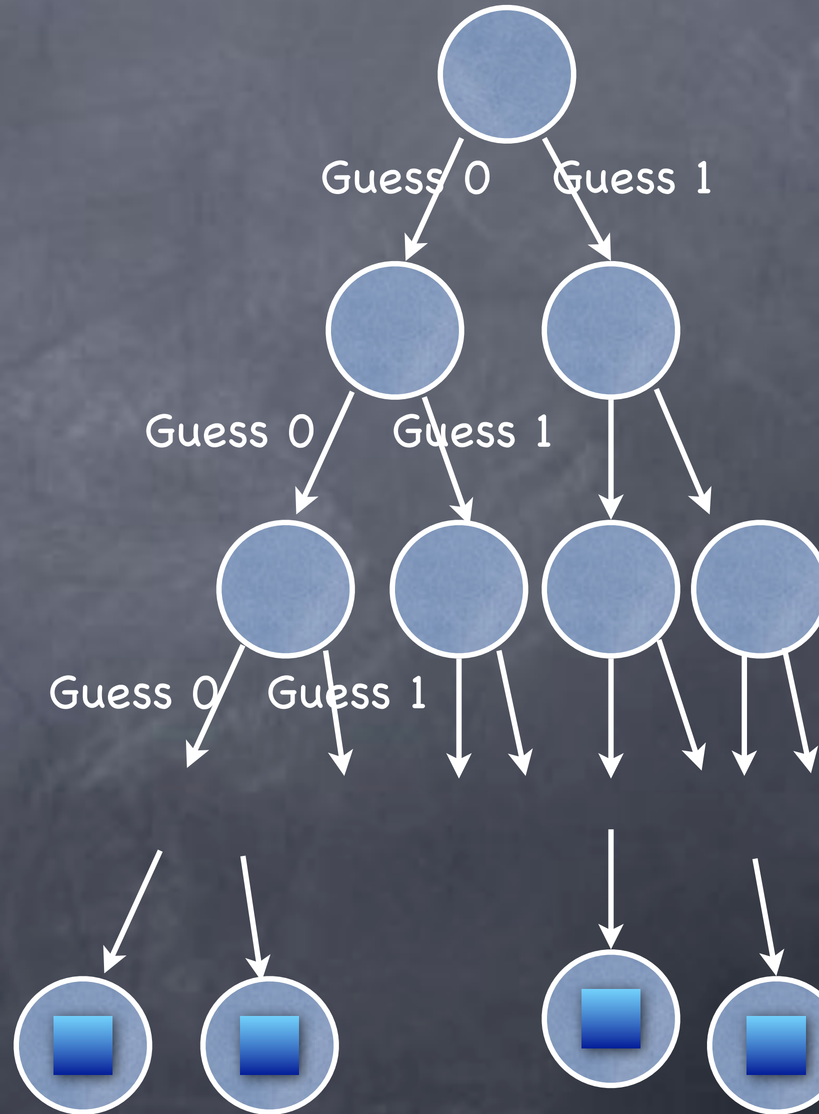
Verification → Non-determinism



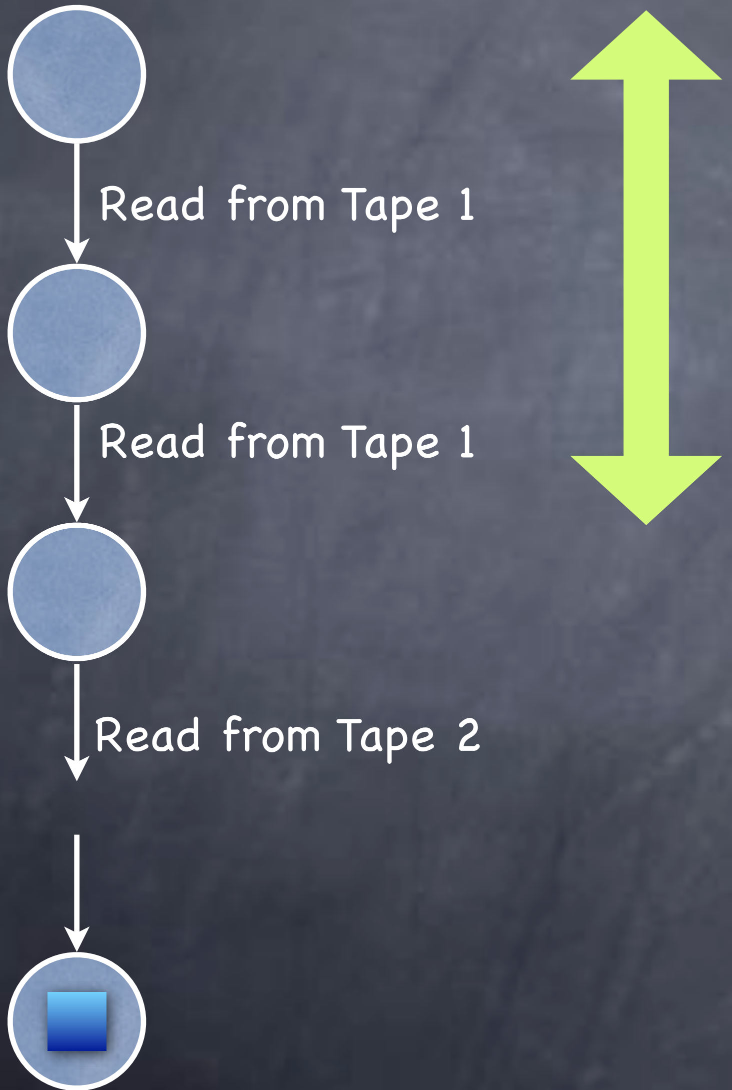


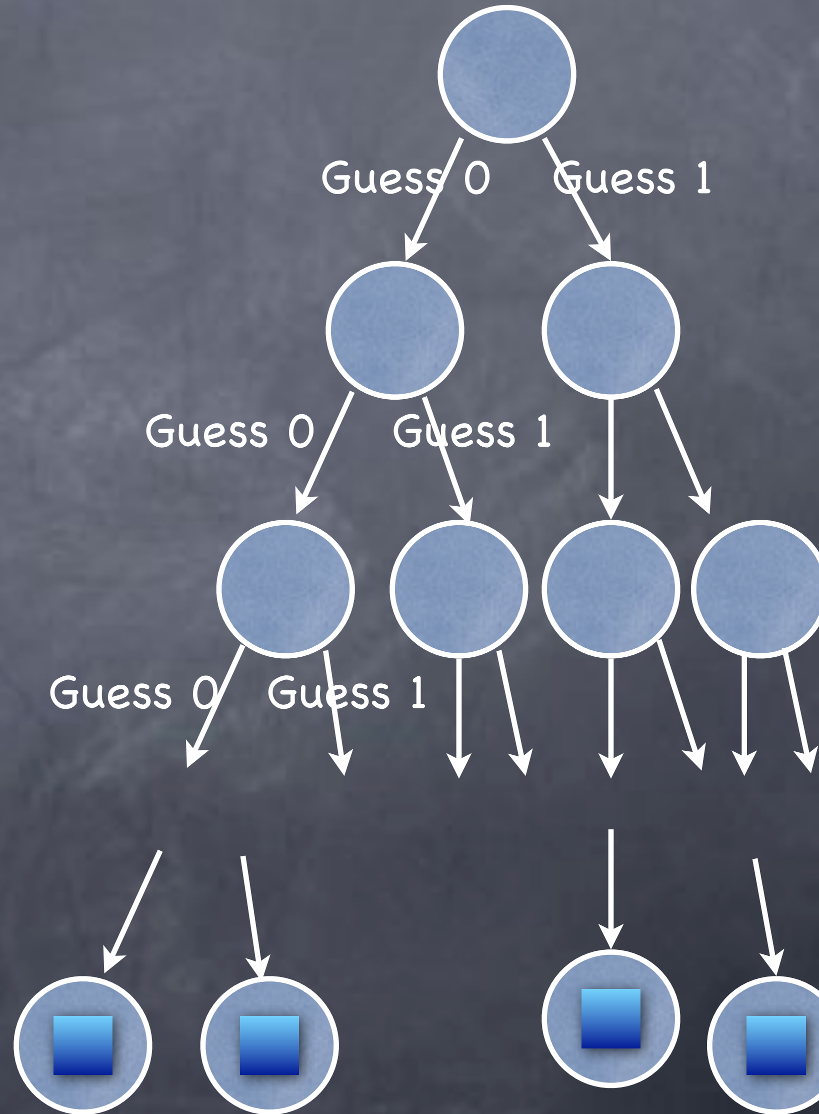
Verification → Non-determinism



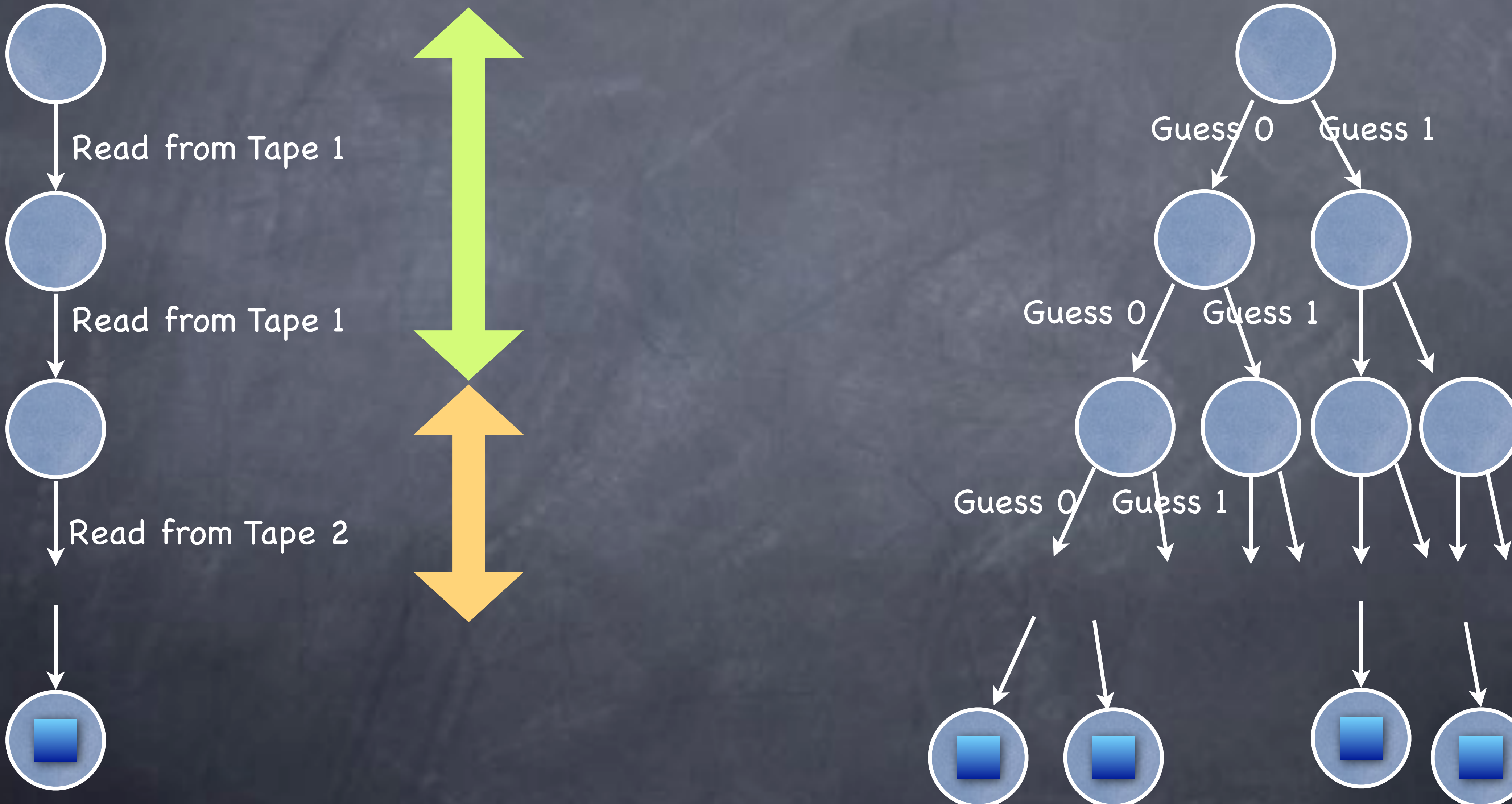


Verification → Non-determinism

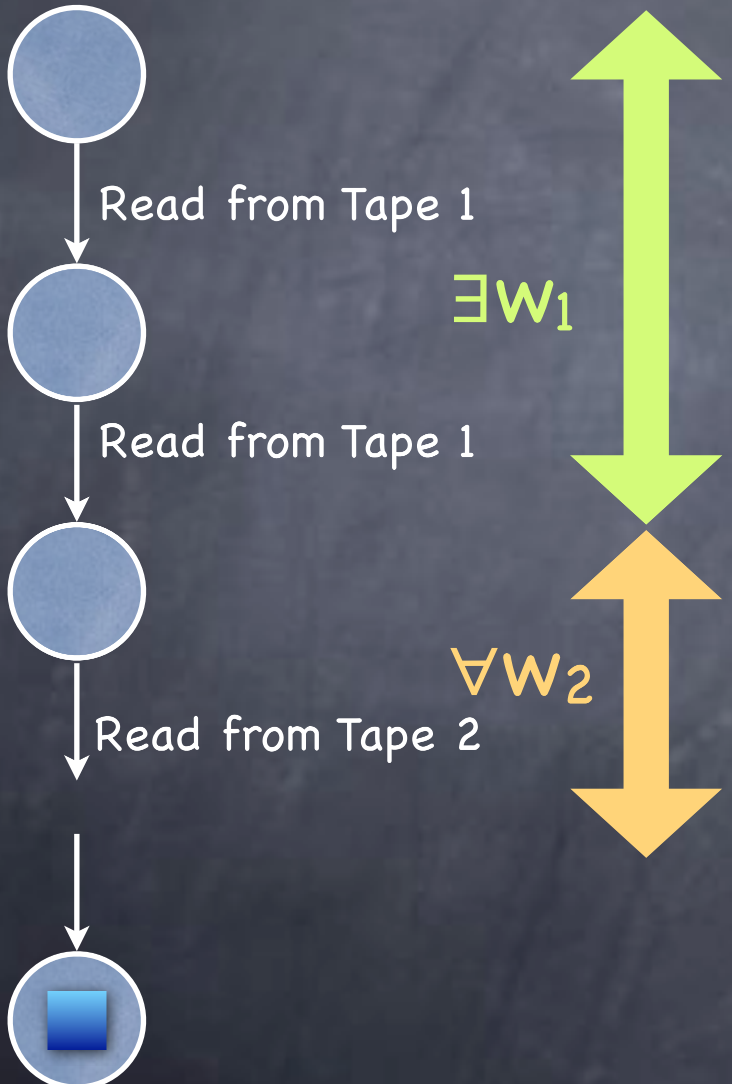


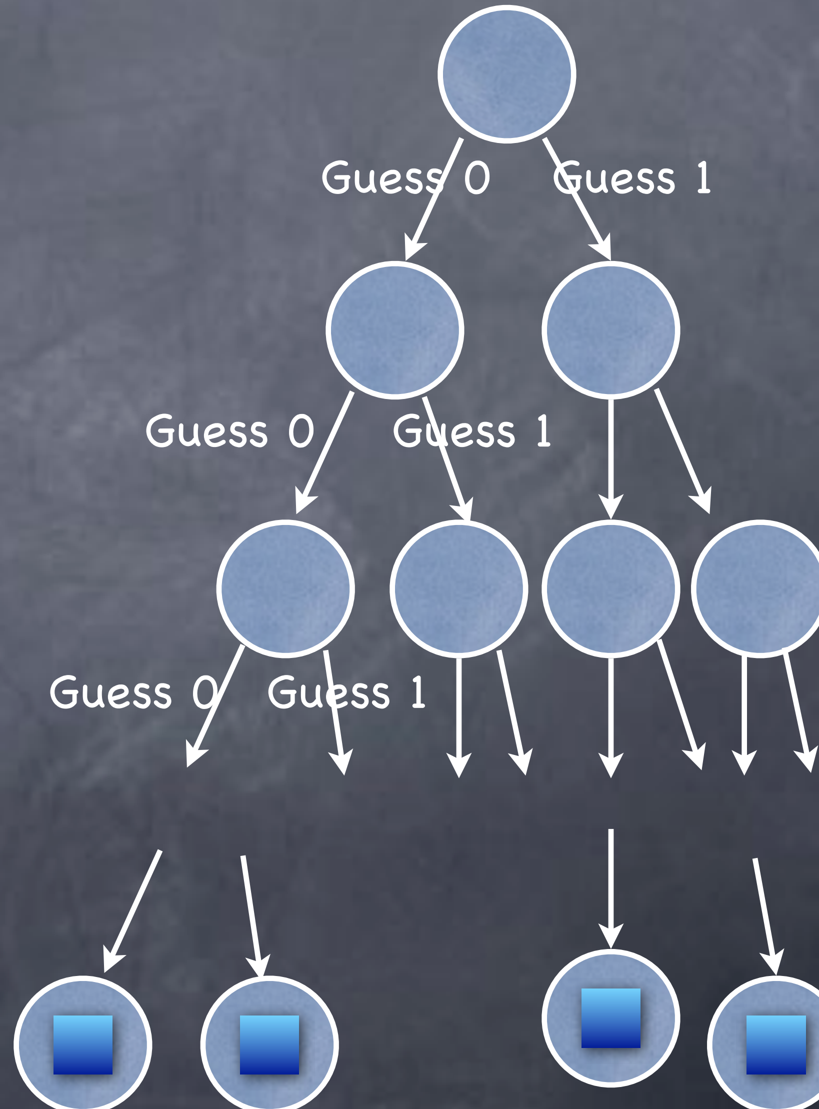


Verification → Non-determinism

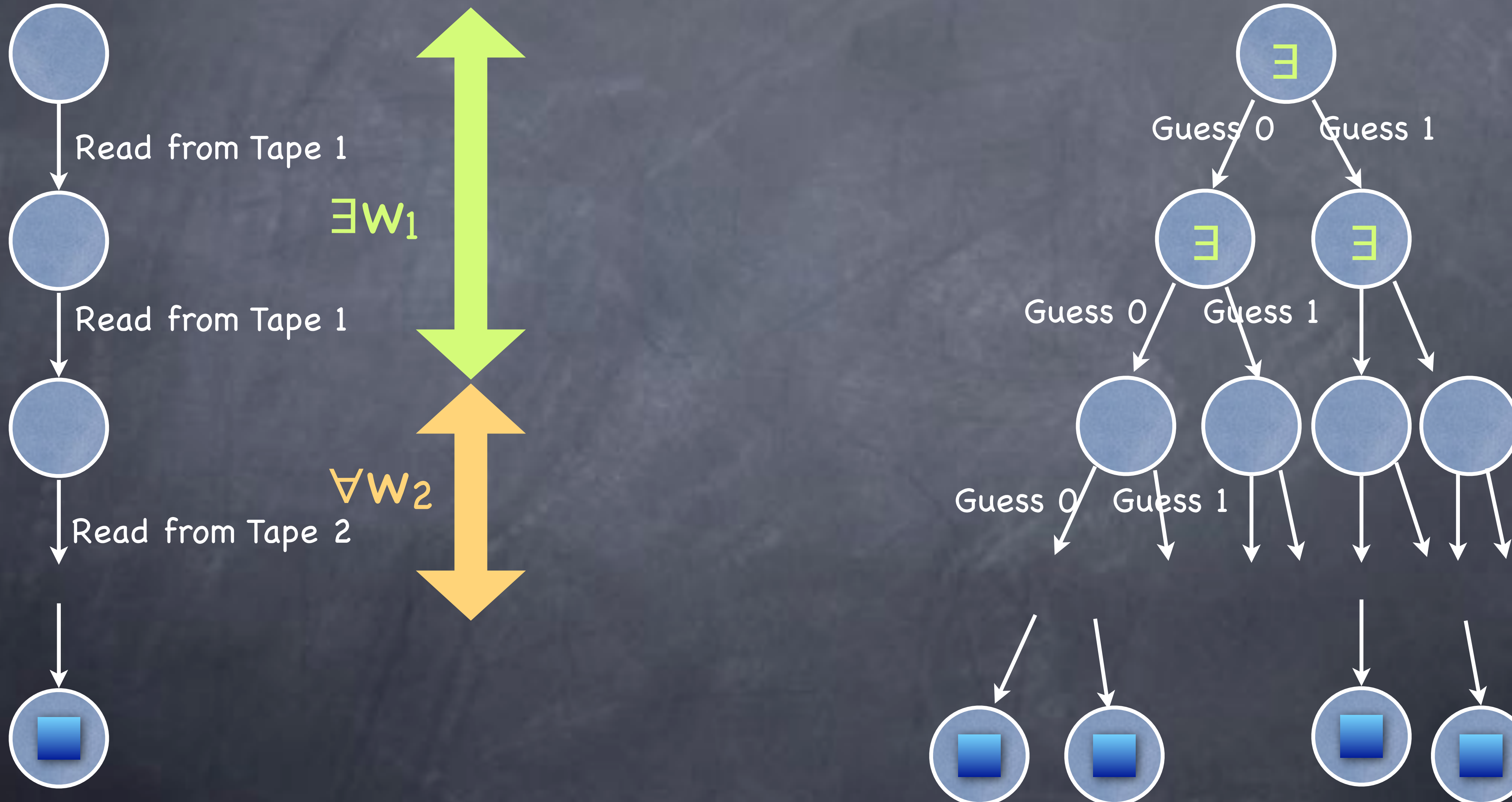


Verification → Non-determinism

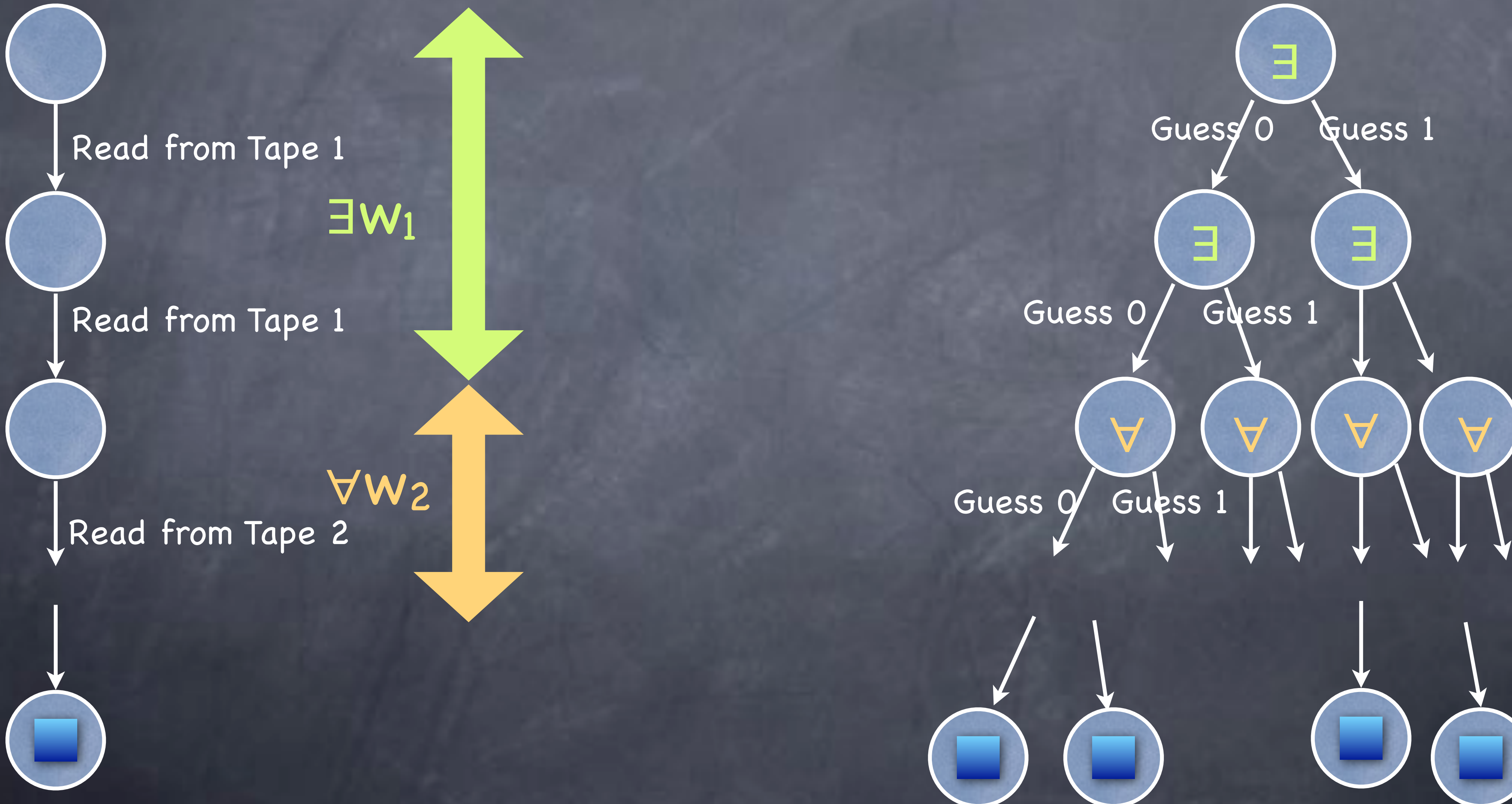




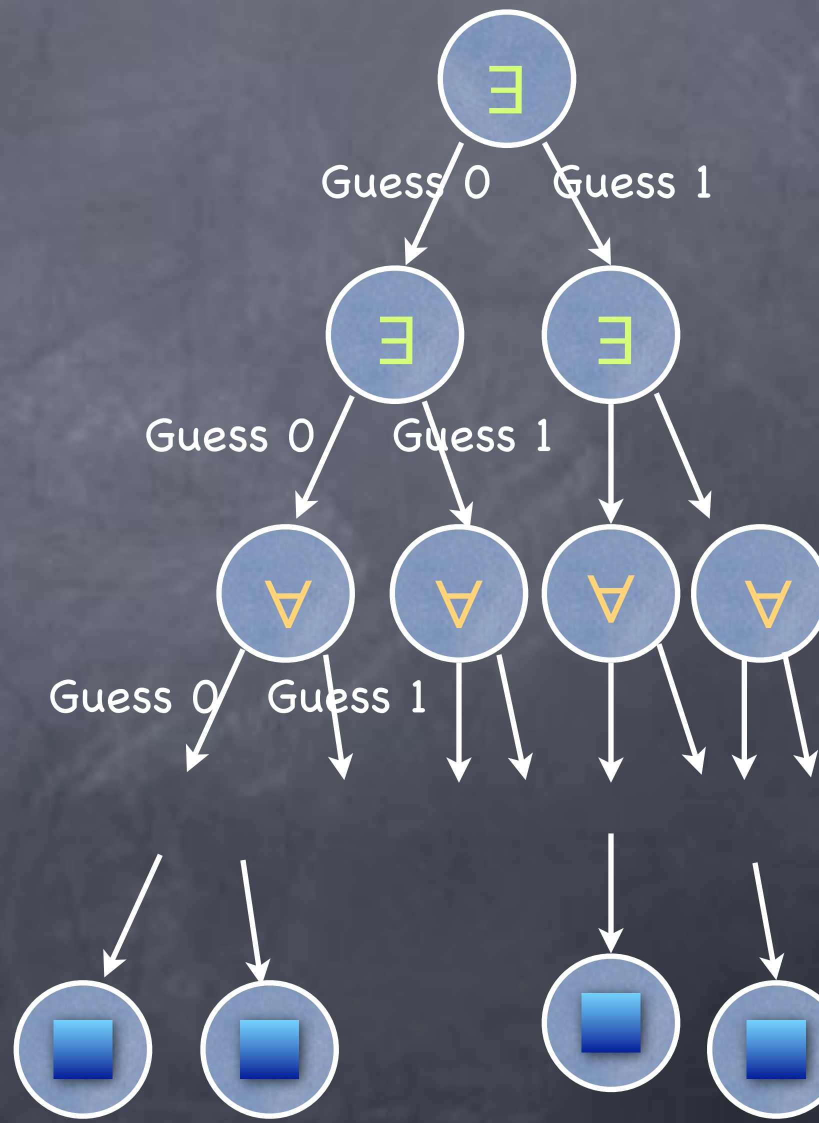
Verification → Non-determinism



Verification → Non-determinism

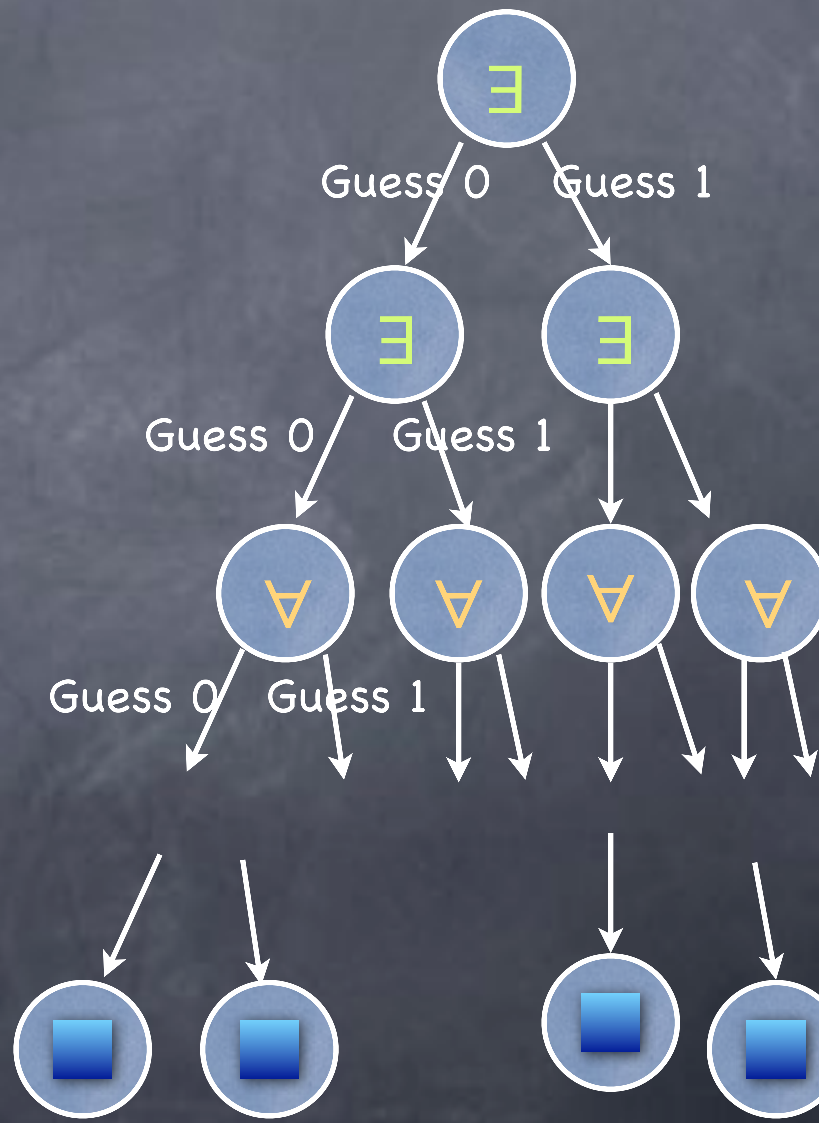


ATM



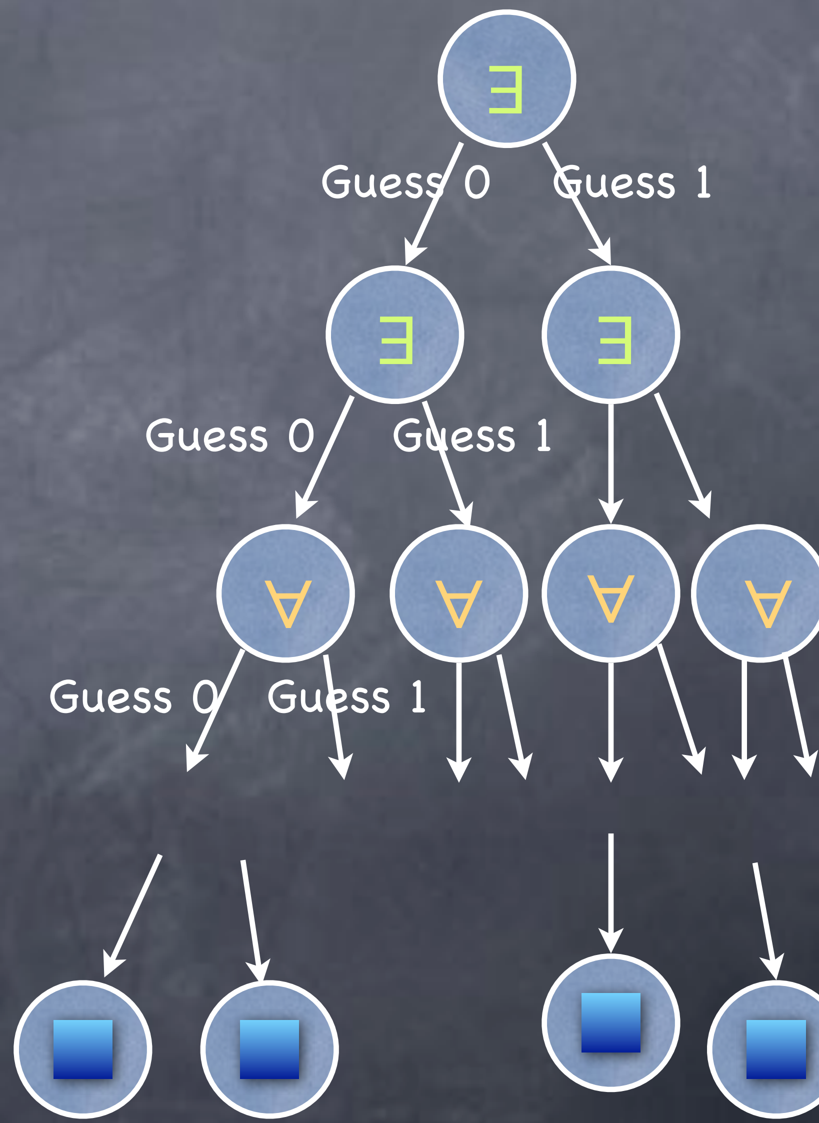
ATM

- Alternating Turing Machine



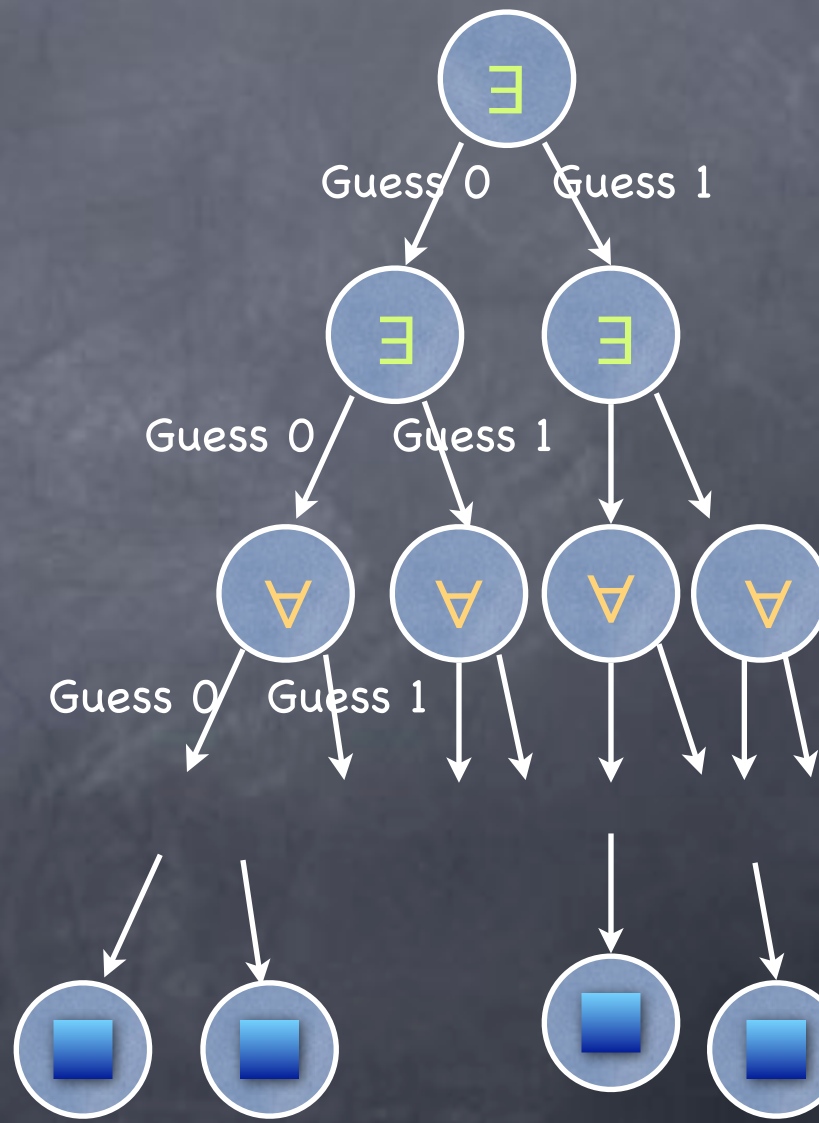
ATM

- Alternating Turing Machine
  - At each step, execution can fork into two



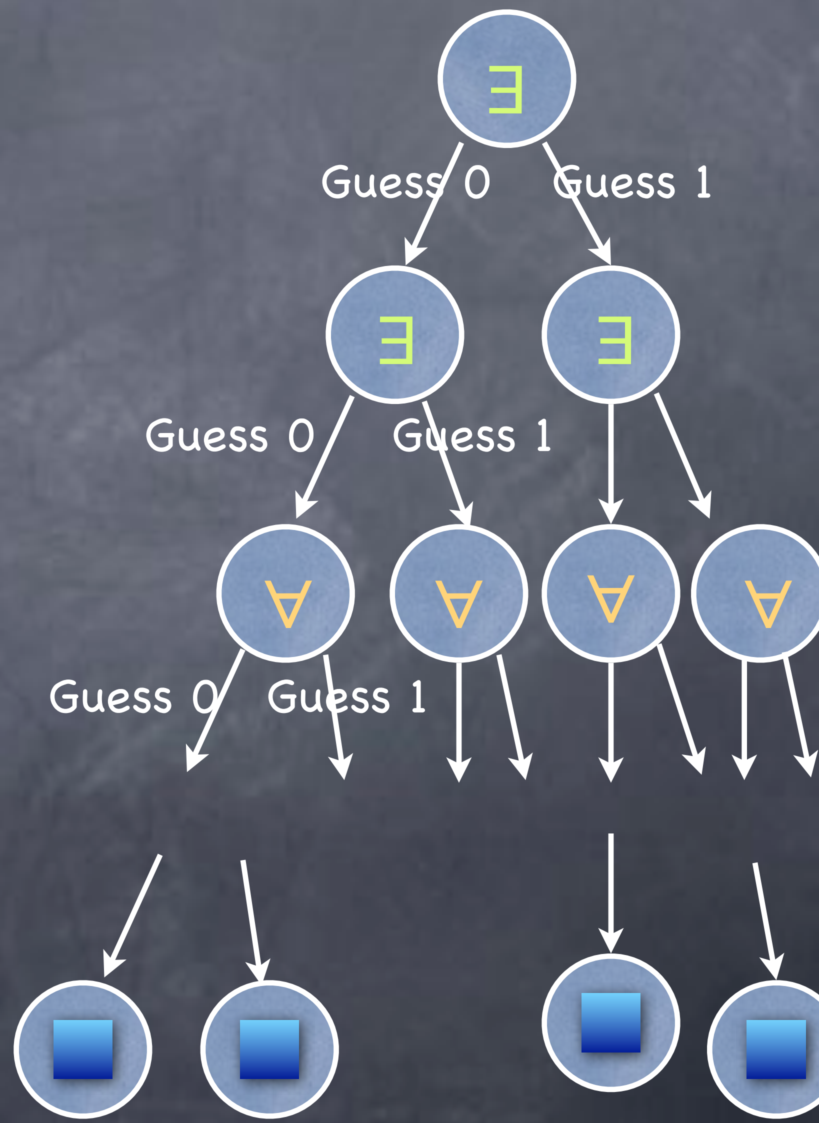
ATM

- Alternating Turing Machine
  - At each step, execution can fork into two
    - Exactly like an NTM or co-NTM



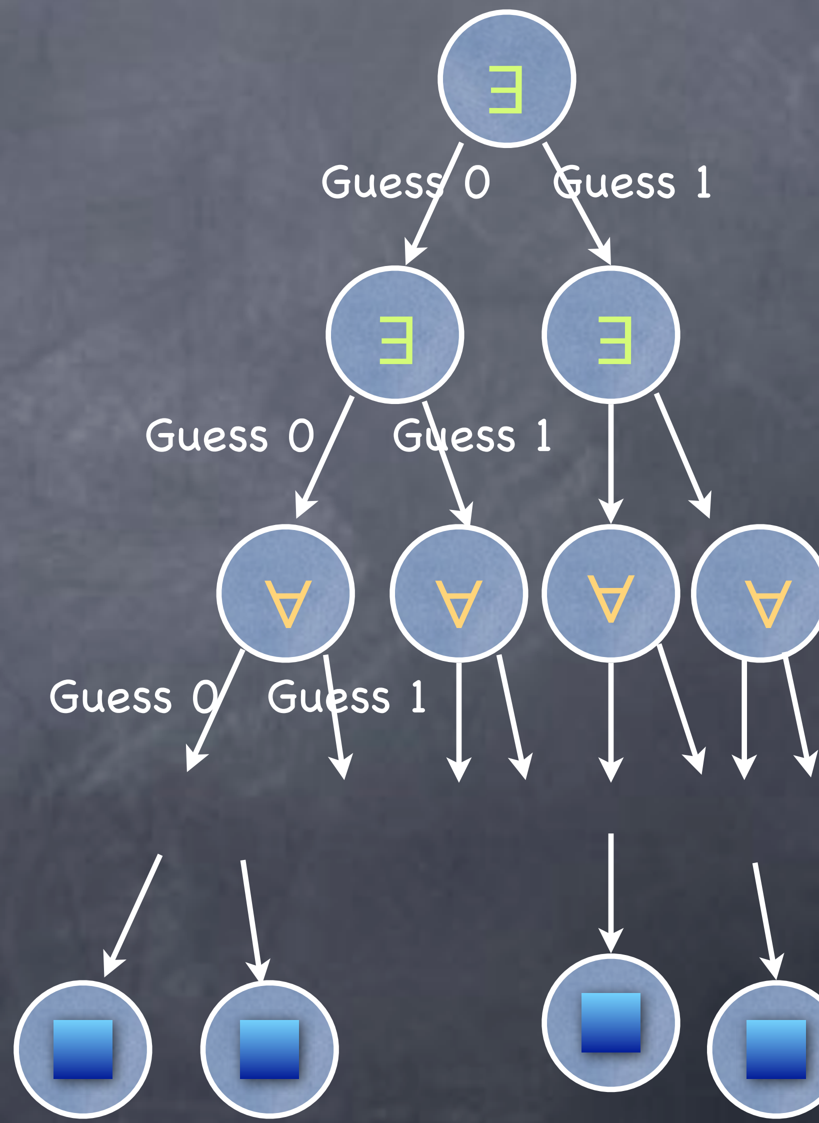
ATM

- Alternating Turing Machine
  - At each step, execution can fork into two
    - Exactly like an NTM or co-NTM
  - Accepting rule is more complex

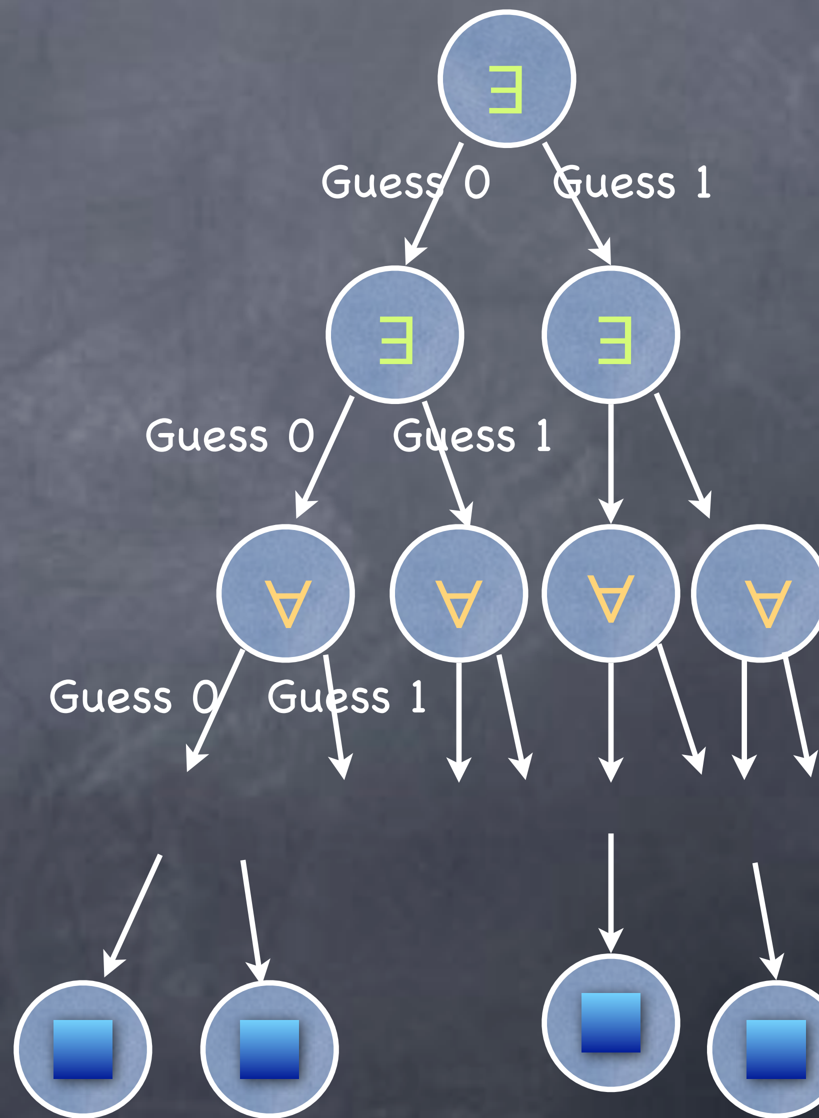


ATM

- Alternating Turing Machine
  - At each step, execution can fork into two
    - Exactly like an NTM or co-NTM
  - Accepting rule is more complex
    - Like in the game tree for QBF

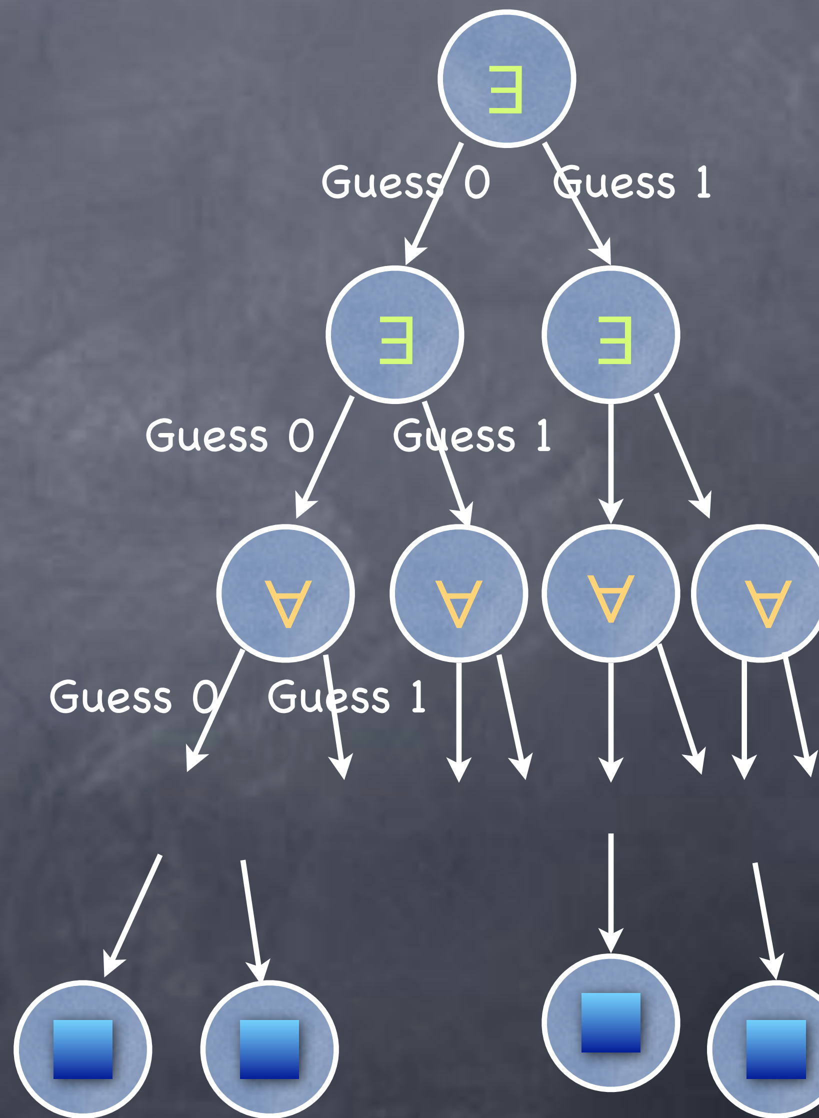


ATM



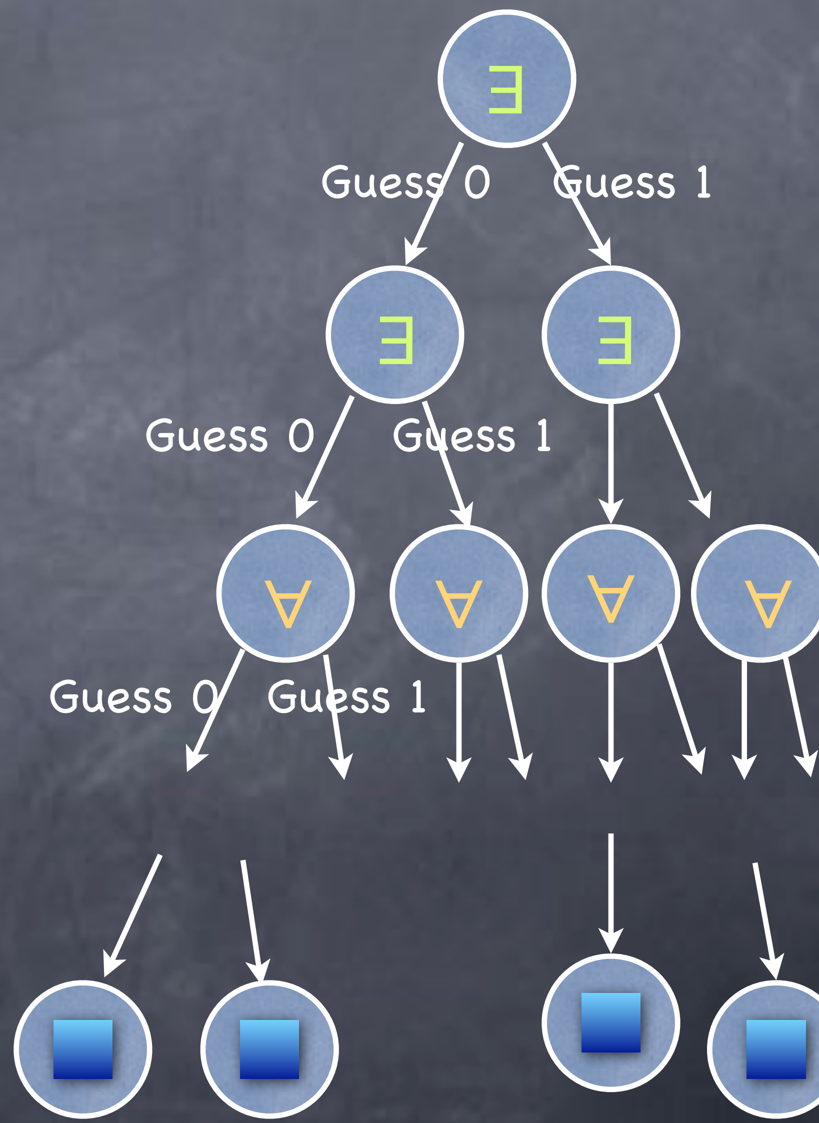
ATM

- Two kinds of configurations: ∃ and ∀



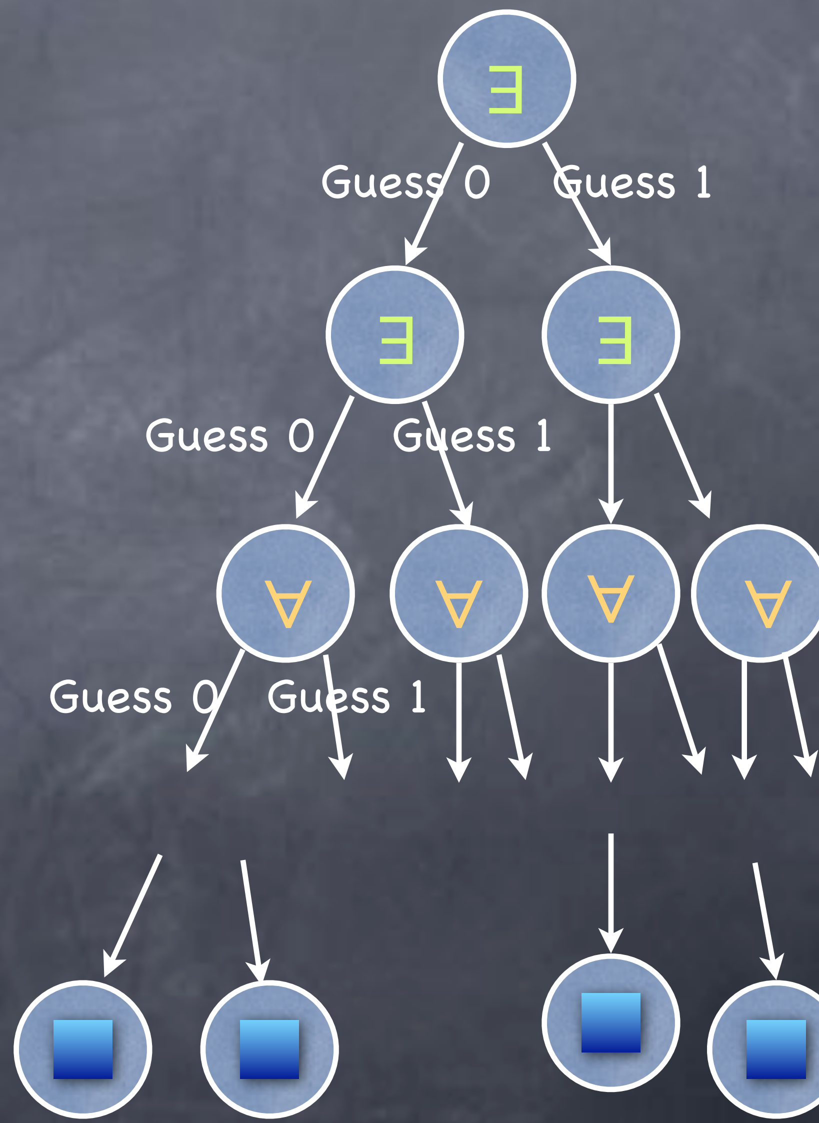
ATM

- Two kinds of configurations: ∃ and ∀
  - Depending on the state



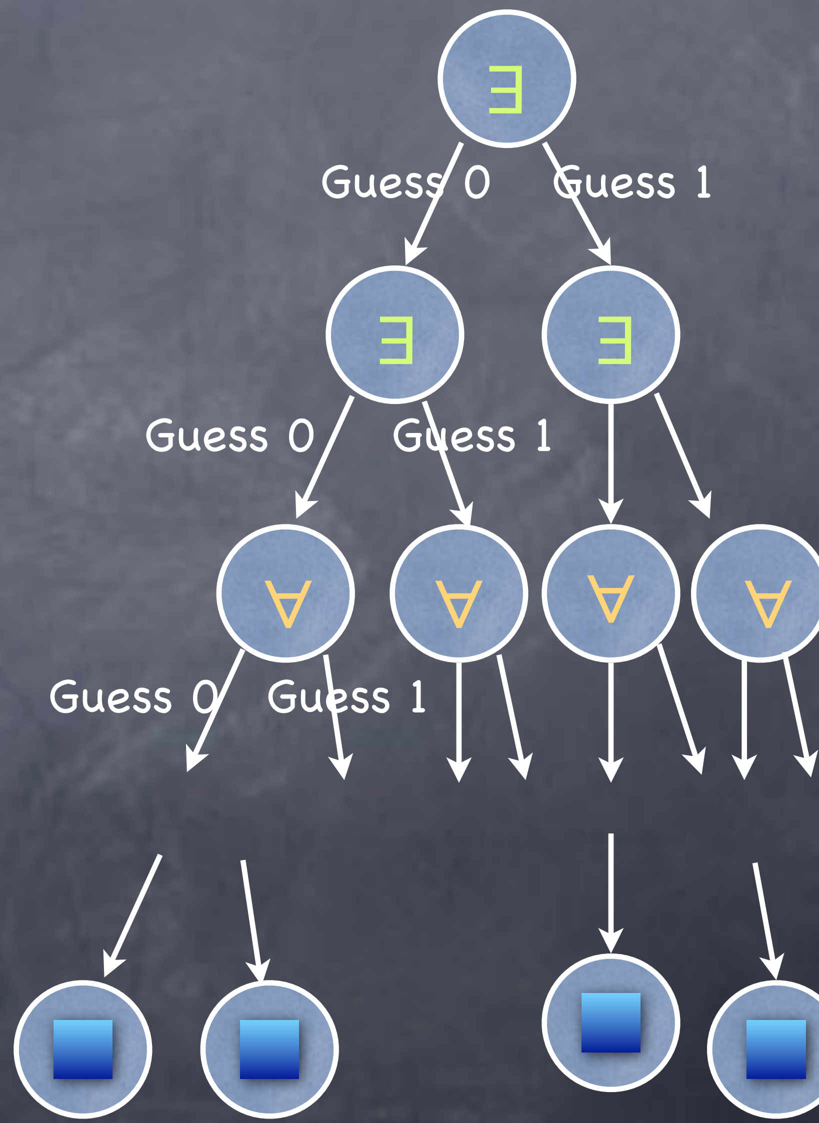
ATM

- Two kinds of configurations: ∃ and ∀
  - Depending on the state
  - A ∃ configuration is accepting if either child is accepting

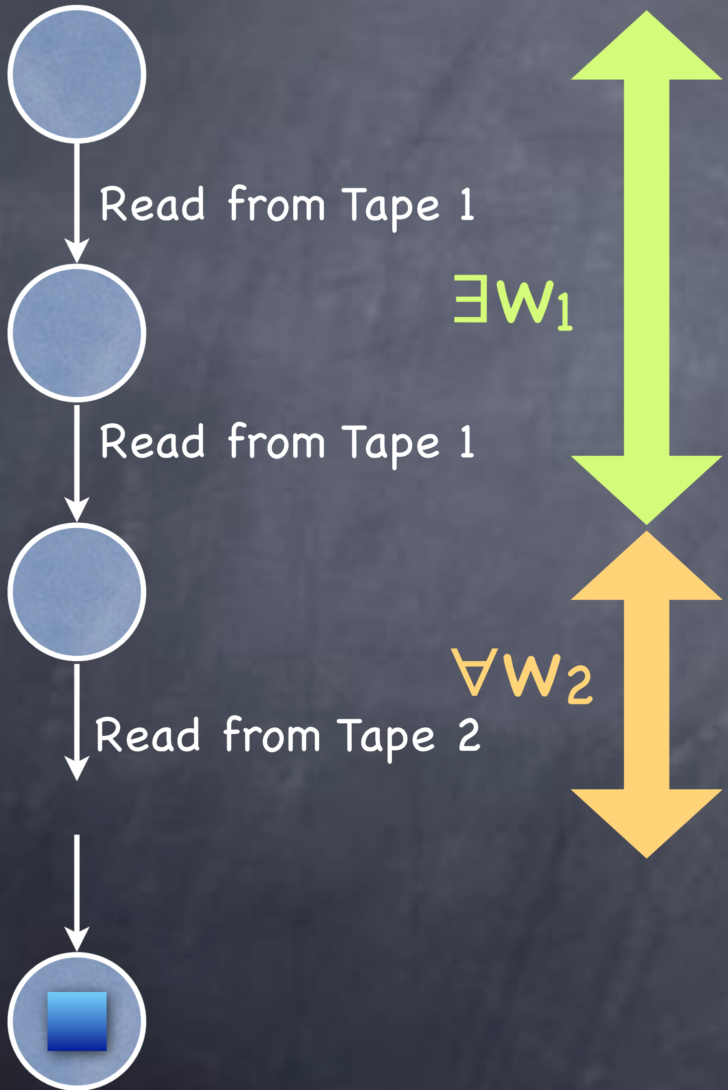


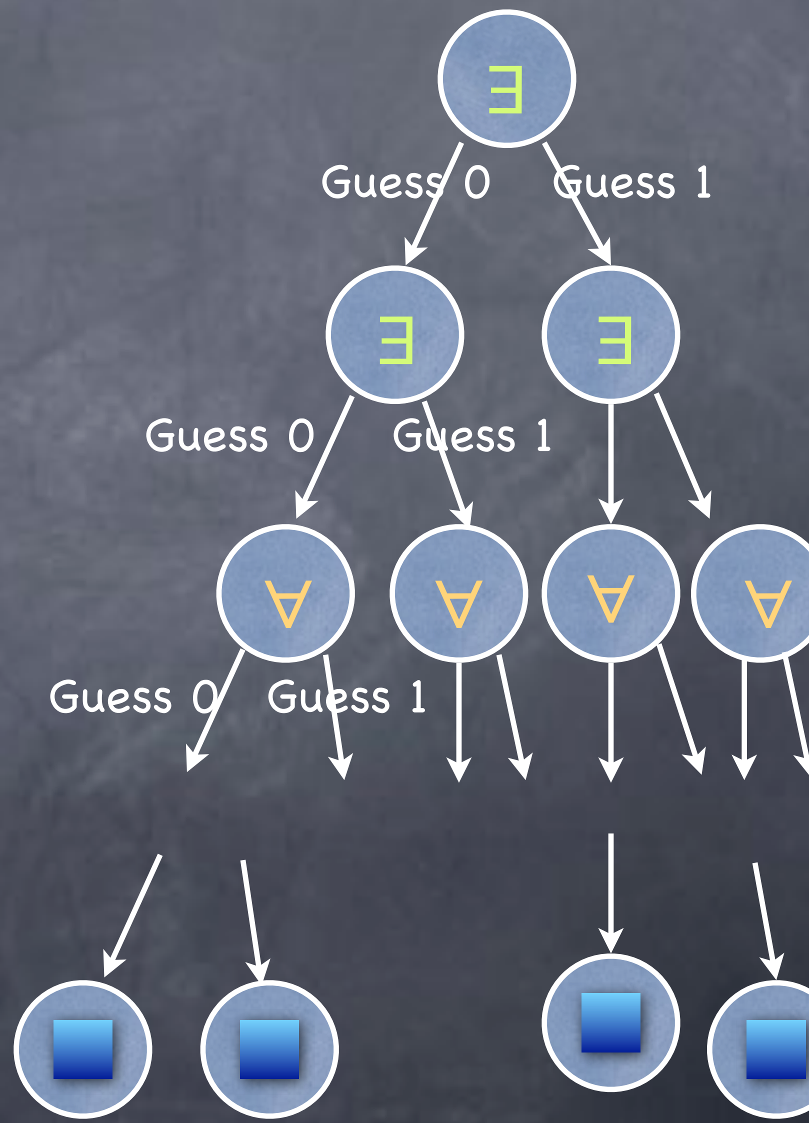
ATM

- Two kinds of configurations: ∃ and ∀
  - Depending on the state
  - A ∃ configuration is accepting if either child is accepting
  - A ∀ configuration is accepting only if both children are accepting

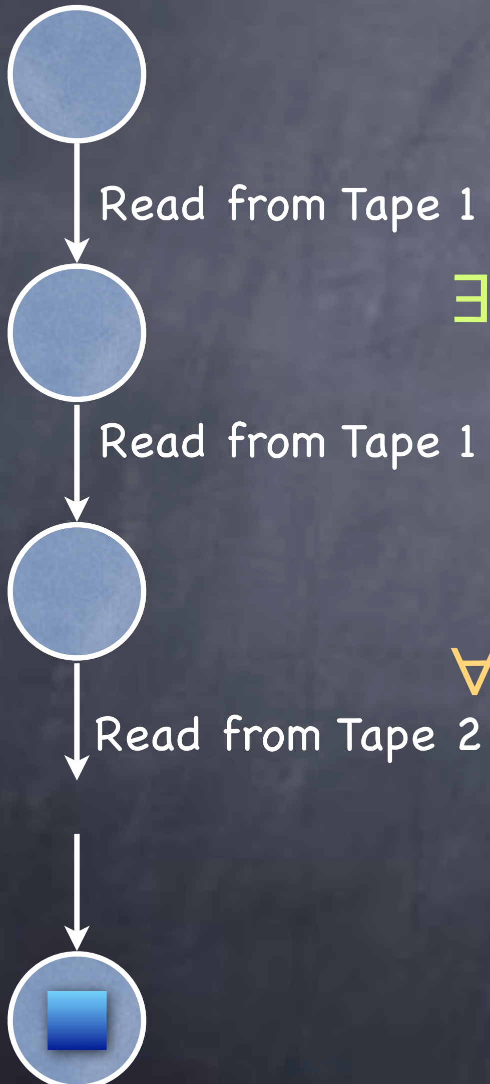


Verification → Non-determinism

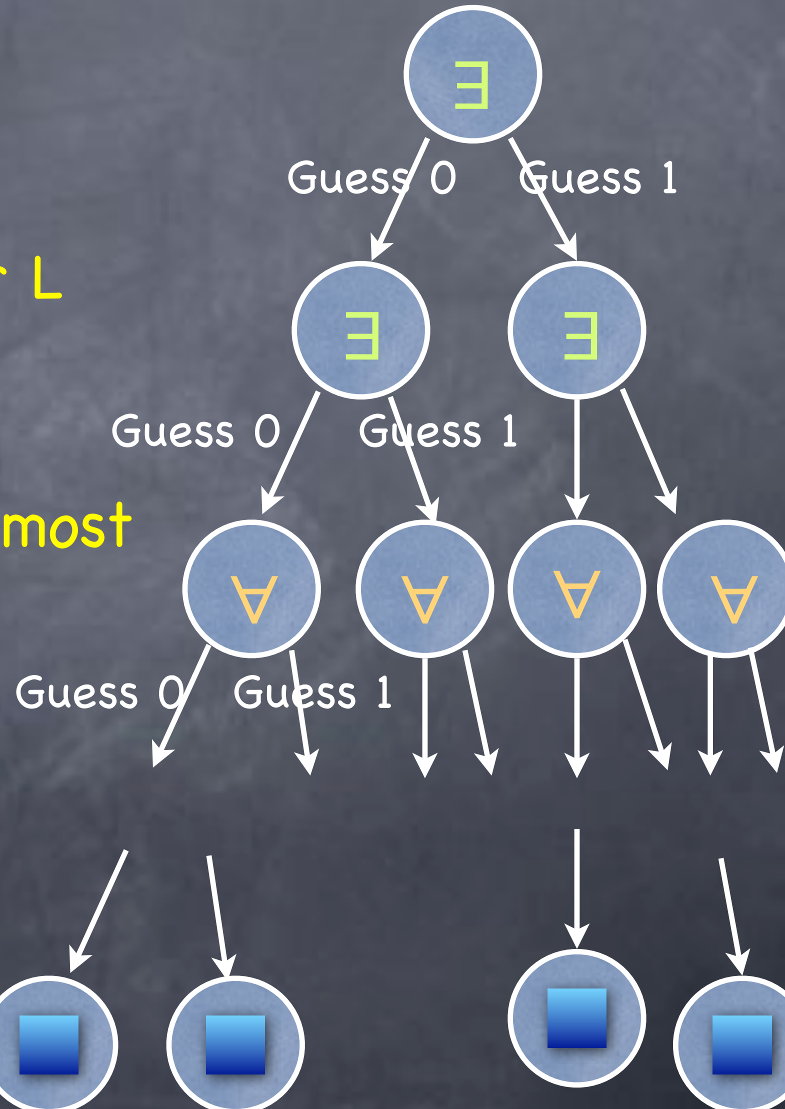




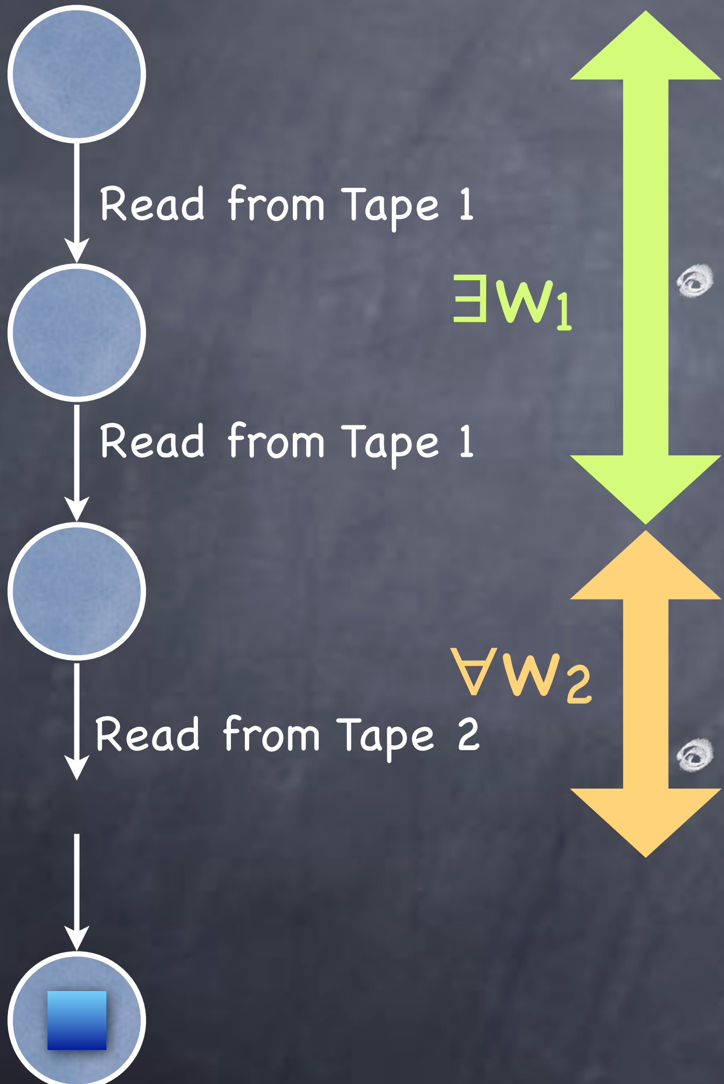
Verification → Non-determinism



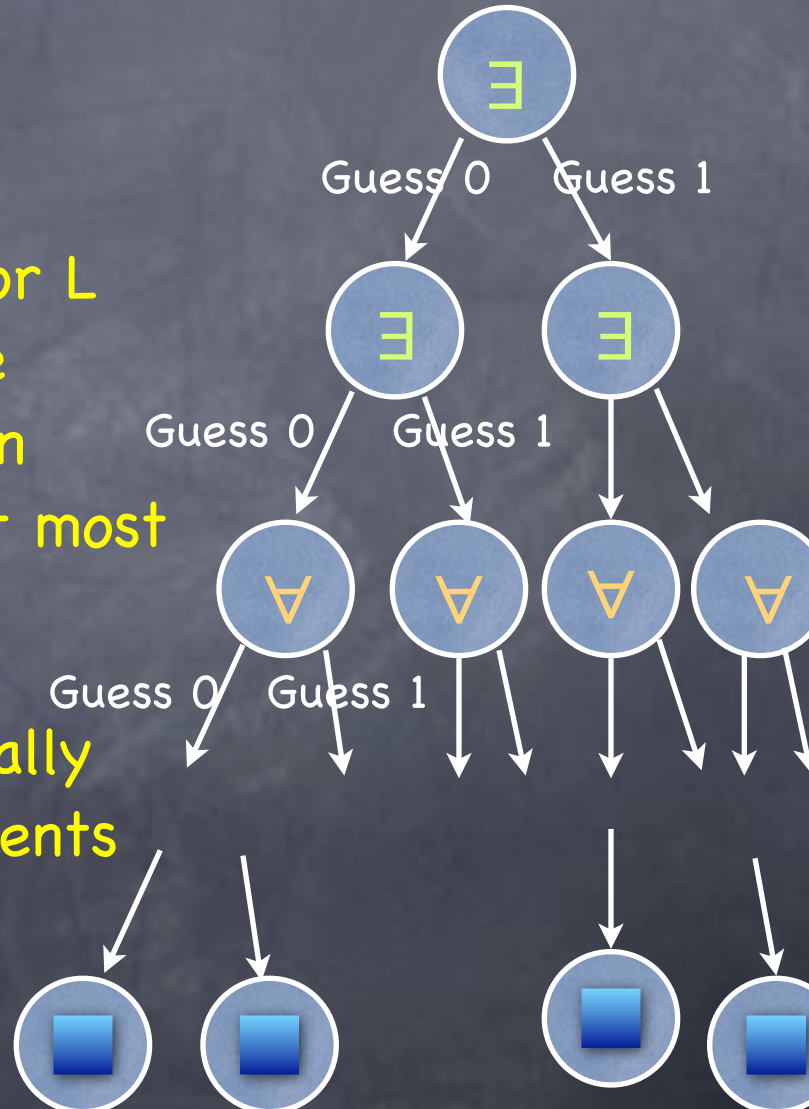
- Given a verifier for L using k certificate tapes, can build an ATM for L with at most k alternations



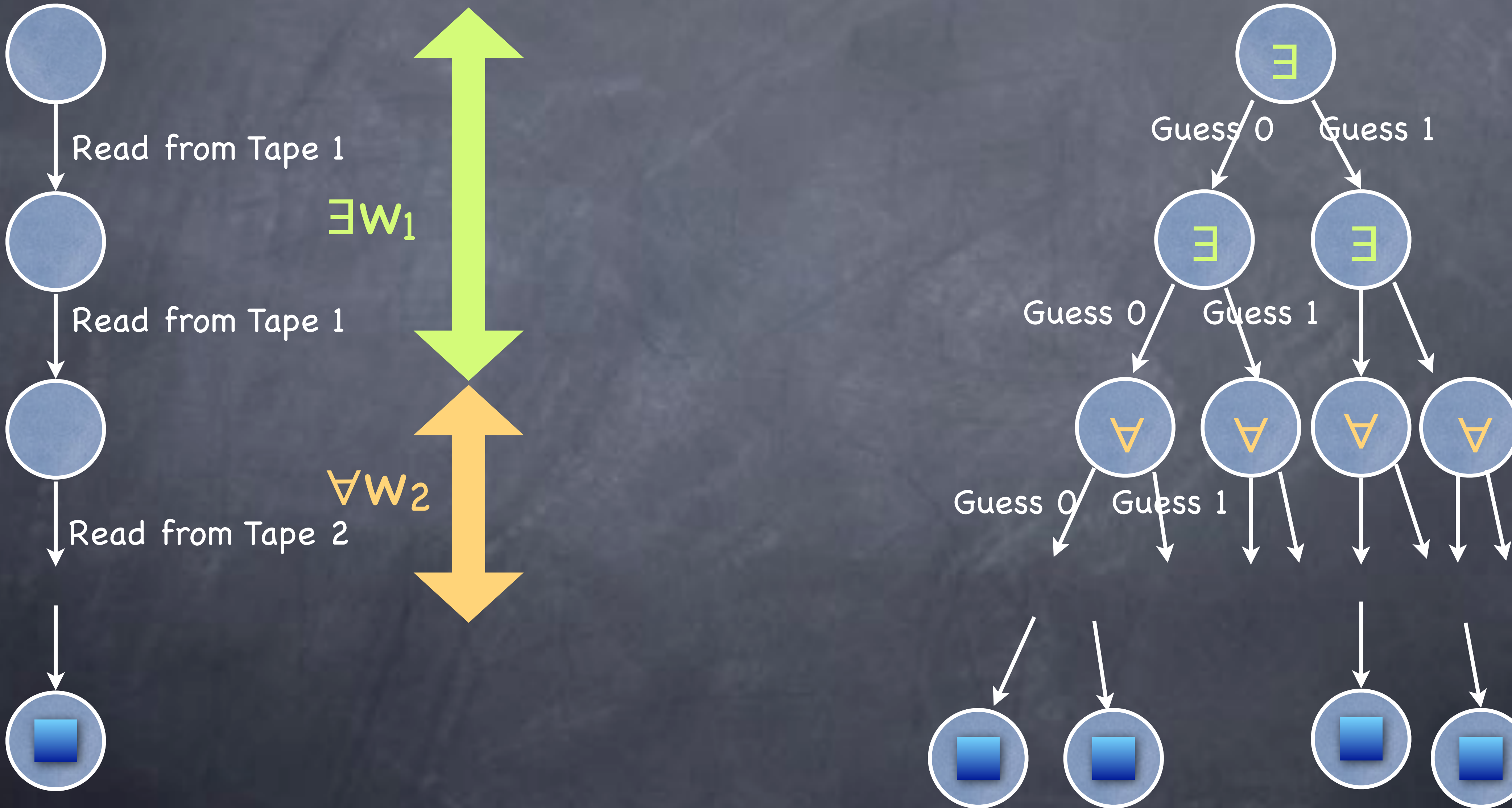
Verification → Non-determinism



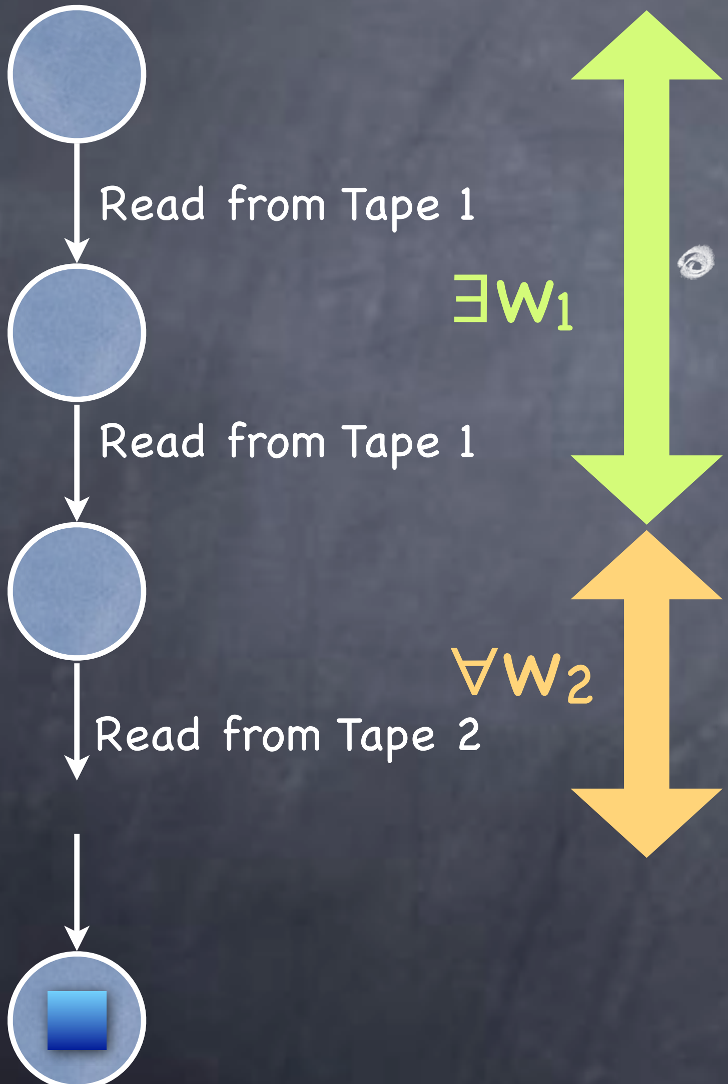
- Given a verifier for L using k certificate tapes, can build an ATM for L with at most k alternations
- Non-deterministically guesses tape contents and runs verifier



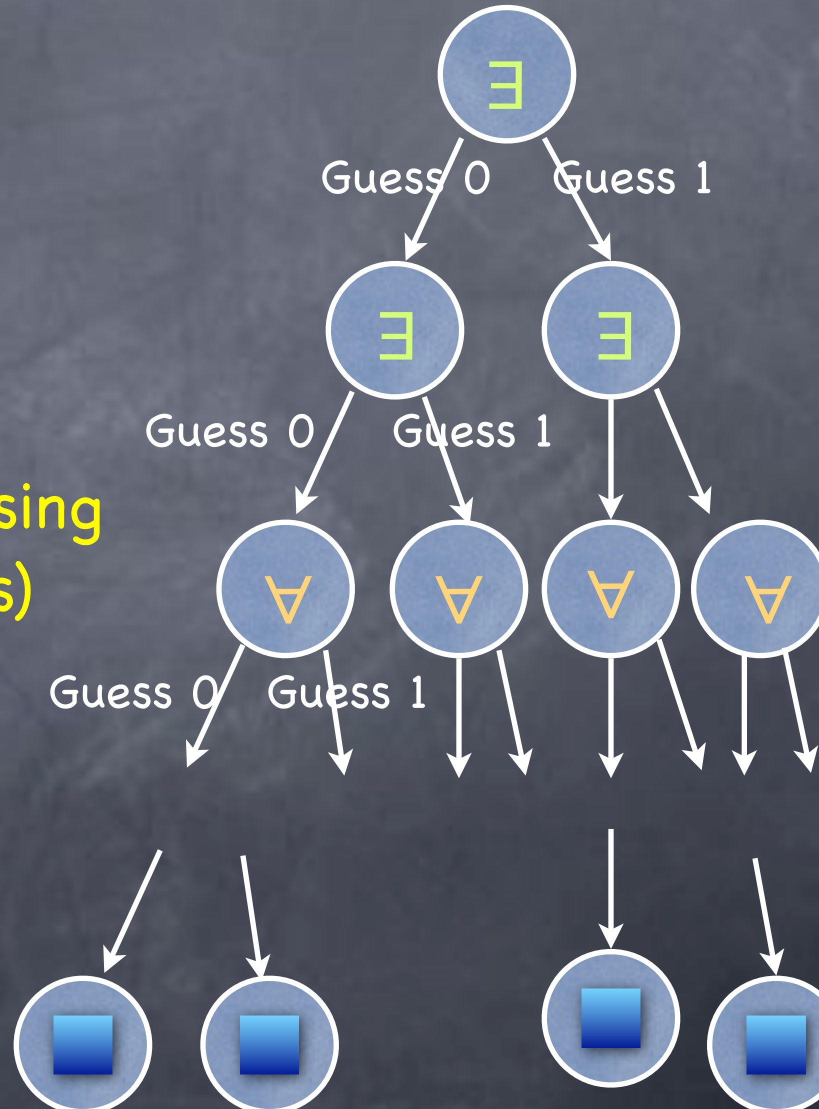
Verification ← Non-determinism



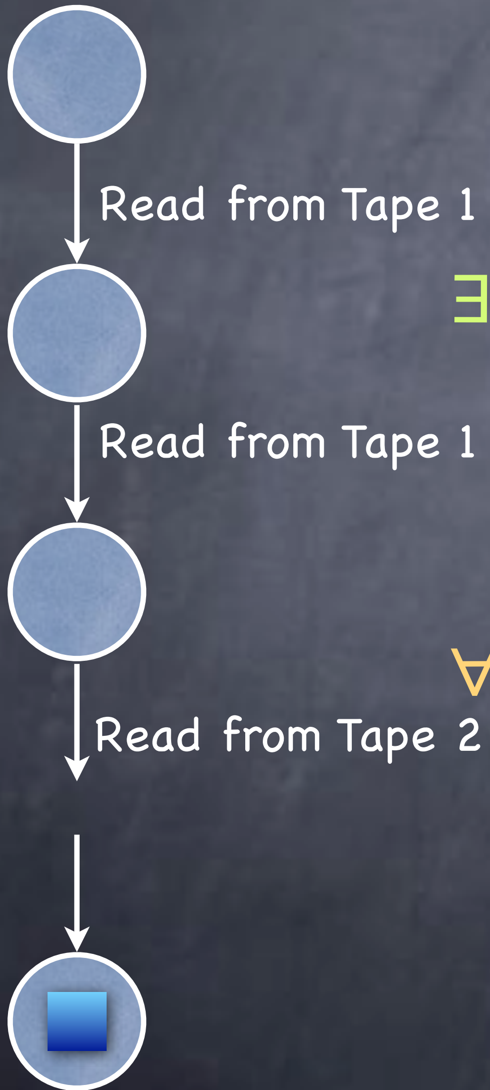
Verification ← Non-determinism



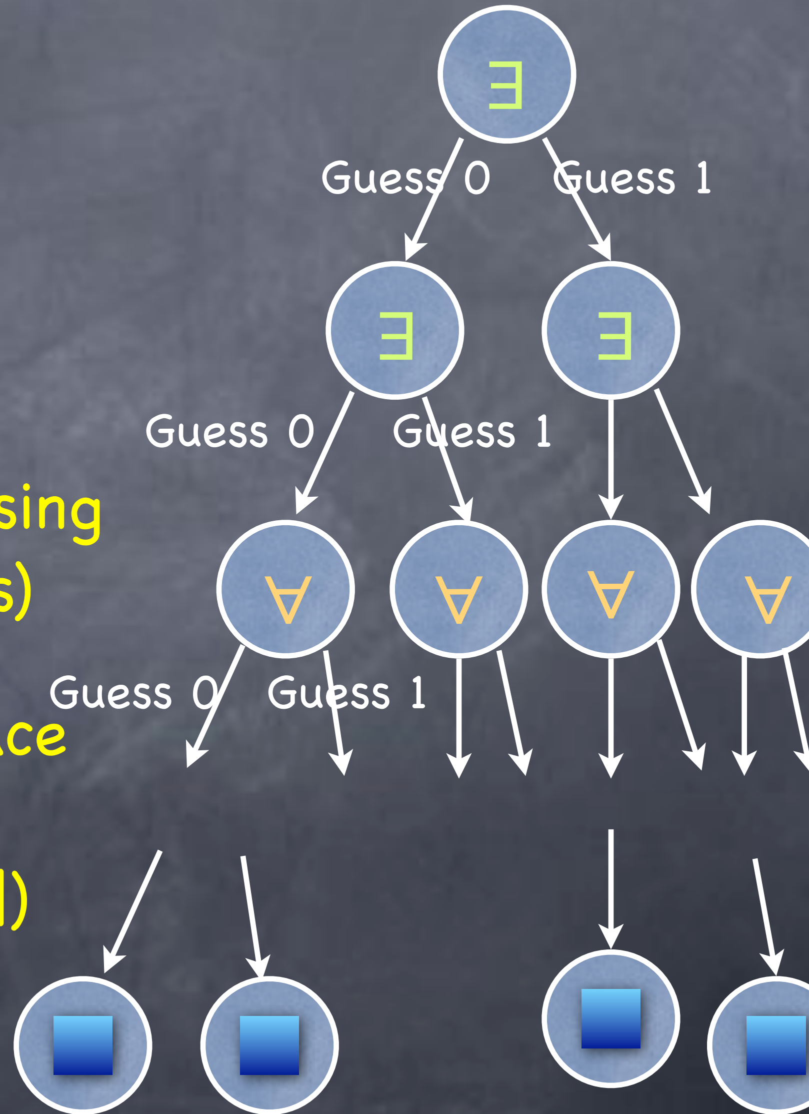
- Given ATM for L with at most k alternations, can build a verifier (using k certificate tapes)



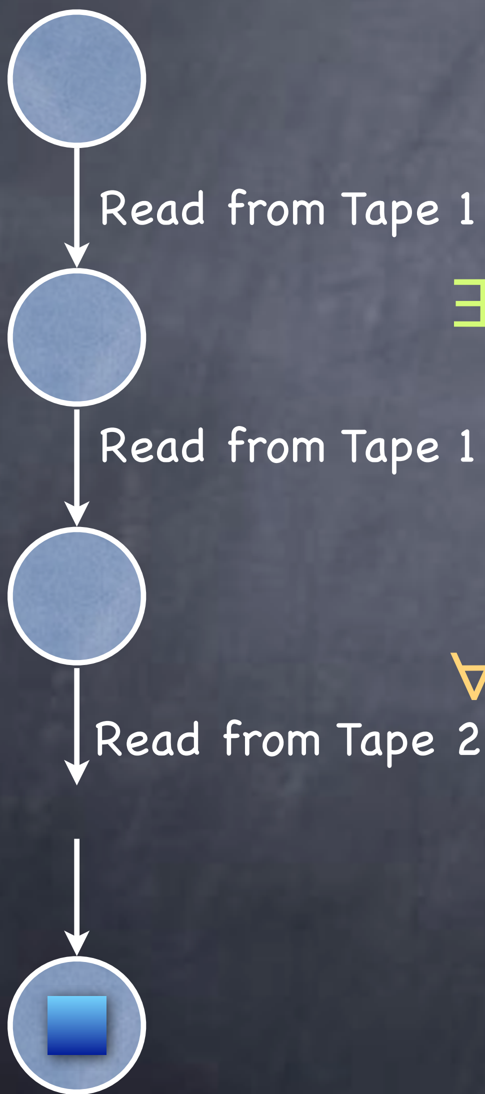
Verification ← Non-determinism



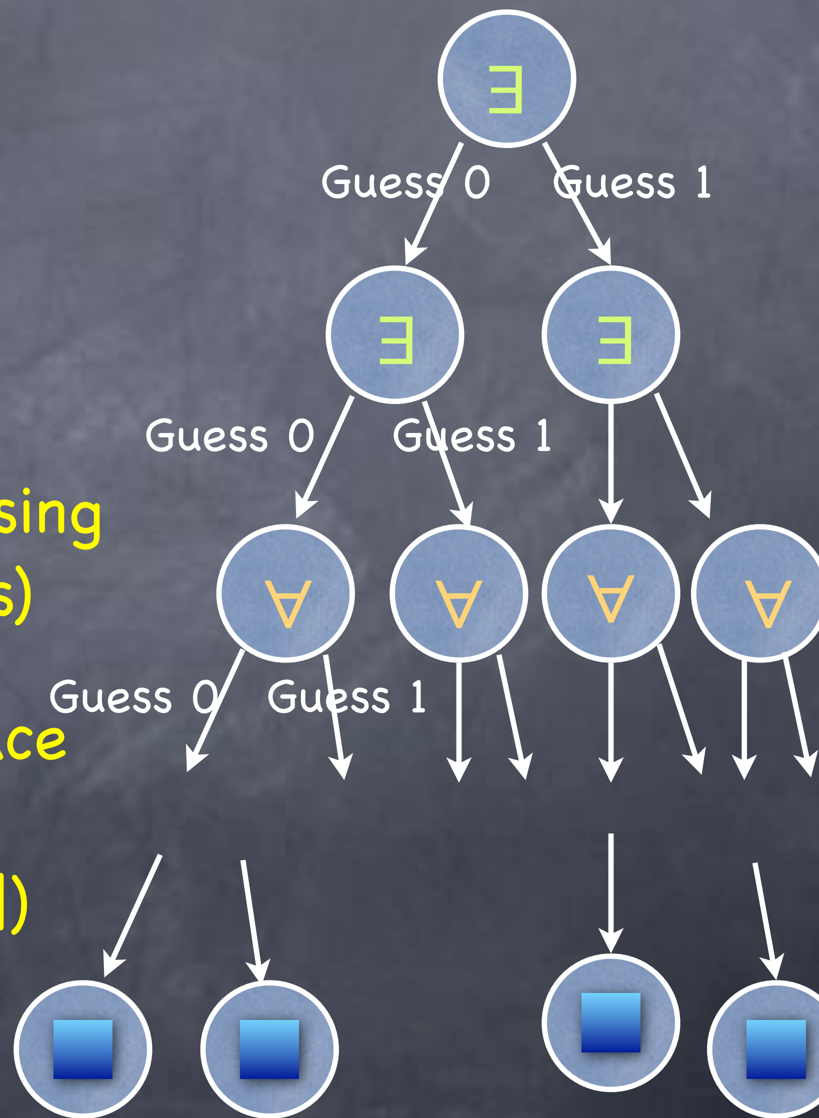
- Given ATM for L with at most k alternations, can build a verifier (using k certificate tapes)
- Same time/space requirements (in terms of |x|)



Verification ← Non-determinism



- Given ATM for L with at most k alternations, can build a verifier (using k certificate tapes)
- Same time/space requirements (in terms of |x|)
- $|w_i|$ = #choices



Time, Space, Alternations

Time, Space, Alternations

- Complexity measures

Time, Space, Alternations

- Complexity measures
  - Time: Maximum number of steps in any thread

Time, Space, Alternations

- Complexity measures
  - Time: Maximum number of steps in any thread
  - Space: Maximum space in any configuration reached

Time, Space, Alternations

- Complexity measures
  - Time: Maximum number of steps in any thread
  - Space: Maximum space in any configuration reached
  - Alternations: Maximum number of quantifier switches in any thread

ATIME

ATIME

- Σ_kTIME, Π_kTIME

ATIME

- Σ_kTIME, Π_kTIME
  - Σ_kTIME(T): languages decided by ATMs with at most k alternations starting with ∃, in time T(n)

ATIME

- Σ_kTIME, Π_kTIME
  - Σ_kTIME(T): languages decided by ATMs with at most k alternations starting with ∃, in time T(n)
  - Σ_kTIME(poly) = Σ_k^P

ATIME

- Σ_kTIME, Π_kTIME
  - Σ_kTIME(T): languages decided by ATMs with at most k alternations starting with ∃, in time T(n)
  - Σ_kTIME(poly) = Σ_k^P
    - Latter being exactly the certificate version

ATIME

- Σ_kTIME, Π_kTIME
  - Σ_kTIME(T): languages decided by ATMs with at most k alternations starting with ∃, in time T(n)
  - Σ_kTIME(poly) = Σ_k^P
    - Latter being exactly the certificate version
- ATIME

ATIME

- Σ_kTIME, Π_kTIME
  - Σ_kTIME(T): languages decided by ATMs with at most k alternations starting with ∃, in time T(n)
  - Σ_kTIME(poly) = Σ_k^P
    - Latter being exactly the certificate version
- ATIME
  - ATIME(T): languages decided by ATMs in time T(n)

ATIME vs. DSPACE

ATIME vs. DSPACE

- ATIME(T) ⊆ DSPACE(T^2)

ATIME vs. DSPACE

- ATIME(T) ⊆ DSPACE(T^2)
  - c.f. NTIME(T) ⊆ DSPACE(T)

ATIME vs. DSPACE

- ATIME(T) ⊆ DSPACE(T^2)
  - c.f. NTIME(T) ⊆ DSPACE(T)
  - AP ⊆ PSPACE

ATIME vs. DSPACE

- ATIME(T) ⊆ DSPACE(T^2)
  - c.f. NTIME(T) ⊆ DSPACE(T)
  - AP ⊆ PSPACE
- But PSPACE ⊆ AP

ATIME vs. DSPACE

- ATIME(T) ⊆ DSPACE(T^2)
  - c.f. NTIME(T) ⊆ DSPACE(T)
  - AP ⊆ PSPACE
- But PSPACE ⊆ AP
  - TQBF in AP (why?)

ATIME vs. DSPACE

- ATIME(T) ⊆ DSPACE(T^2)
  - c.f. NTIME(T) ⊆ DSPACE(T)
  - AP ⊆ PSPACE
- But PSPACE ⊆ AP
  - TQBF in AP (why?)
- AP = PSPACE

ATIME(T) $\subseteq$ DSPACE(T^2)

ATIME(T) $\subseteq$ DSPACE(T^2)

- Evaluate if the start configuration is accepting, recursively

ATIME(T) $\subseteq$ DSPACE(T^2)

- Evaluate if the start configuration is accepting, recursively
  - A ∃ configuration is accepting if any child is, and a ∀ configuration is accepting if all children are

ATIME(T) $\subseteq$ DSPACE(T^2)

- Evaluate if the start configuration is accepting, recursively
  - A $\exists$ configuration is accepting if any child is, and a $\forall$ configuration is accepting if all children are
- Space needed: depth x size of configuration

ATIME(T) ⊆ DSPACE(T^2)

- Evaluate if the start configuration is accepting, recursively
  - A ∃ configuration is accepting if any child is, and a ∀ configuration is accepting if all children are
- Space needed: depth x size of configuration
  - Depth = # alternations = O(T). Also, size of configuration = O(T) as any thread runs for time O(T)

ATIME(T) ⊆ DSPACE(T^2)

- Evaluate if the start configuration is accepting, recursively
  - A ∃ configuration is accepting if any child is, and a ∀ configuration is accepting if all children are
- Space needed: depth x size of configuration
  - Depth = # alternations = O(T). Also, size of configuration = O(T) as any thread runs for time O(T)
  - O(T^2)

ASPACE vs. DTIME

ASPACE vs. DTIME

- $\text{ASPACE}(S) = \text{DTIME}(2^{O(S)})$

ASPACE vs. DTIME

- ASPACE(S) = DTIME($2^{O(S)}$)
    - Recall, already seen NSPACE(S) $\subseteq$ DTIME($2^{O(S)}$)

ASPACE vs. DTIME

- ASPACE(S) = DTIME($2^{O(S)}$)
  - Recall, already seen NSPACE(S) $\subseteq$ DTIME($2^{O(S)}$)
    - Poly-time connectivity in configuration graph of size at most $2^{O(S)}$

ASPACE vs. DTIME

- ASPACE(S) = DTIME($2^{O(S)}$)
  - Recall, already seen NSPACE(S) ⊆ DTIME($2^{O(S)}$)
    - Poly-time connectivity in configuration graph of size at most $2^{O(S)}$
    - Instead of connectivity, can recursively label all accepting nodes (2 lookups per node: in poly(S) time). So ASPACE(S) ⊆ DTIME($2^{O(S)}$)

ASPACE vs. DTIME

- $ASPACE(S) = DTIME(2^{O(S)})$
  - Recall, already seen $NSPACE(S) \subseteq DTIME(2^{O(S)})$
    - Poly-time connectivity in configuration graph of size at most $2^{O(S)}$
    - Instead of connectivity, can recursively label all accepting nodes (2 lookups per node: in poly(S) time). So $ASPACE(S) \subseteq DTIME(2^{O(S)})$
  - To show $DTIME(2^{O(S)}) \subseteq ASPACE(S)$

$\mathrm{DTIME}(2^{O(S)}) \subseteq \mathrm{ASPACE}(S)$

$DTIME(2^{O(S)}) \subseteq ASPACE(S)$

- To decide, is configuration after t steps accepting

$\mathrm{DTIME}(2^{O(S)}) \subseteq \mathrm{ASPACE}(S)$

- To decide, is configuration after t steps accepting
  - Accept configuration, with unique first cell α (blank tape cell and unique accept state)

DTIME($2^{O(S)}$) $\subseteq$ ASPACE(S)

- To decide, is configuration after t steps accepting
  - Accept configuration, with unique first cell α (blank tape cell and unique accept state)
    - Once there, stays there

$DTIME(2^{O(S)}) \subseteq ASPACE(S)$

- To decide, is configuration after t steps accepting
  - Accept configuration, with unique first cell α (blank tape cell and unique accept state)
    - Once there, stays there
  - Is first cell of config after t steps α

DTIME($2^{O(S)}$) ⊆ ASPACE(S)

- To decide, is configuration after t steps accepting
  - Accept configuration, with unique first cell α (blank tape cell and unique accept state)
    - Once there, stays there
  - Is first cell of config after t steps α
    - C(i,j,x) : if after i steps, j^{th} cell of config is x

DTIME($2^{O(S)}$) $\subseteq$ ASPACE(S)

- To decide, is configuration after t steps accepting
  - Accept configuration, with unique first cell α (blank tape cell and unique accept state)
    - Once there, stays there
  - Is first cell of config after t steps α
    - $C(i,j,x)$: if after i steps, j^{th} cell of config is x
    - Need to check $C(t,1,\alpha)$

ATM for TM simulation

ATM for TM simulation

- C(i,j,x) : if after i steps, j^{th} cell of config is x

ATM for TM simulation

- C(i,j,x) : if after i steps, j^{th} cell of config is x
  - Recall reduction in Cook's theorem

ATM for TM simulation

- C(i,j,x) : if after i steps, j^{th} cell of config is x
  - Recall reduction in Cook's theorem
    - If C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c) then C(i,j,x) iff x=F(a,b,c)

ATM for TM simulation

- C(i,j,x) : if after i steps, j^{th} cell of config is x
  - Recall reduction in Cook's theorem
    - If C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c) then C(i,j,x) iff x=F(a,b,c)
  - C(i,j,x): ∃a,b,c st x=F(a,b,c) and C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c)

ATM for TM simulation

- C(i,j,x) : if after i steps, j^{th} cell of config is x
  - Recall reduction in Cook's theorem
    - If C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c) then C(i,j,x) iff x=F(a,b,c)
  - C(i,j,x): ∃a,b,c st x=F(a,b,c) and C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c)
    - Base case: C(0,j,x) easy to check from input

ATM for TM simulation

- C(i,j,x) : if after i steps, j^{th} cell of config is x
  - Recall reduction in Cook's theorem
    - If C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c) then C(i,j,x) iff x=F(a,b,c)
  - C(i,j,x): ∃a,b,c st x=F(a,b,c) and C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c)
    - Base case: C(0,j,x) easy to check from input
    - Naive recursion: Extra O(S) space to store i,j at each level for $2^{O(S)}$ levels!

ATM for TM simulation

ATM for TM simulation

- ATM to check if C(i,j,x)

ATM for TM simulation

- ATM to check if C(i,j,x)
  - C(i,j,x): ∃a,b,c st x=F(a,b,c) and C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c)

ATM for TM simulation

- ATM to check if C(i,j,x)
  - C(i,j,x): ∃a,b,c st x=F(a,b,c) and C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c)
  - **Tail-recursion in parallel forks**

ATM for TM simulation

- ATM to check if C(i,j,x)
  - C(i,j,x): ∃a,b,c st x=F(a,b,c) and C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c)
  - **Tail-recursion in parallel forks**
    - Check x=F(a,b,c); then enter universal state, and non-deterministically choose one of the three conditions to check

ATM for TM simulation

- ATM to check if C(i,j,x)
  - C(i,j,x): ∃a,b,c st x=F(a,b,c) and C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c)
  - **Tail-recursion in parallel forks**
    - Check x=F(a,b,c); then enter universal state, and non-deterministically choose one of the three conditions to check
    - Overwrite C(i,j,x) with C(i-1,...) and reuse space

ATM for TM simulation

- ATM to check if C(i,j,x)
  - C(i,j,x): ∃a,b,c st x=F(a,b,c) and C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c)
  - **Tail-recursion in parallel forks**
    - Check x=F(a,b,c); then enter universal state, and non-deterministically choose one of the three conditions to check
    - Overwrite C(i,j,x) with C(i-1,...) and reuse space
    - Stay within the same O(S) space at each level!

ATM for TM simulation

- ATM to check if C(i,j,x)
  - C(i,j,x): ∃a,b,c st x=F(a,b,c) and C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c)
  - **Tail-recursion in parallel forks**

    Gets the AND check for free. No need to use a stack.
    - Check x=F(a,b,c); then enter universal state, and non-deterministically choose one of the three conditions to check
    - Overwrite C(i,j,x) with C(i-1,...) and reuse space
    - Stay within the same O(S) space at each level!

ASPACE vs. DTIME

ASPACE vs. DTIME

- ASPACE(S) = DTIME($2^{O(S)}$)

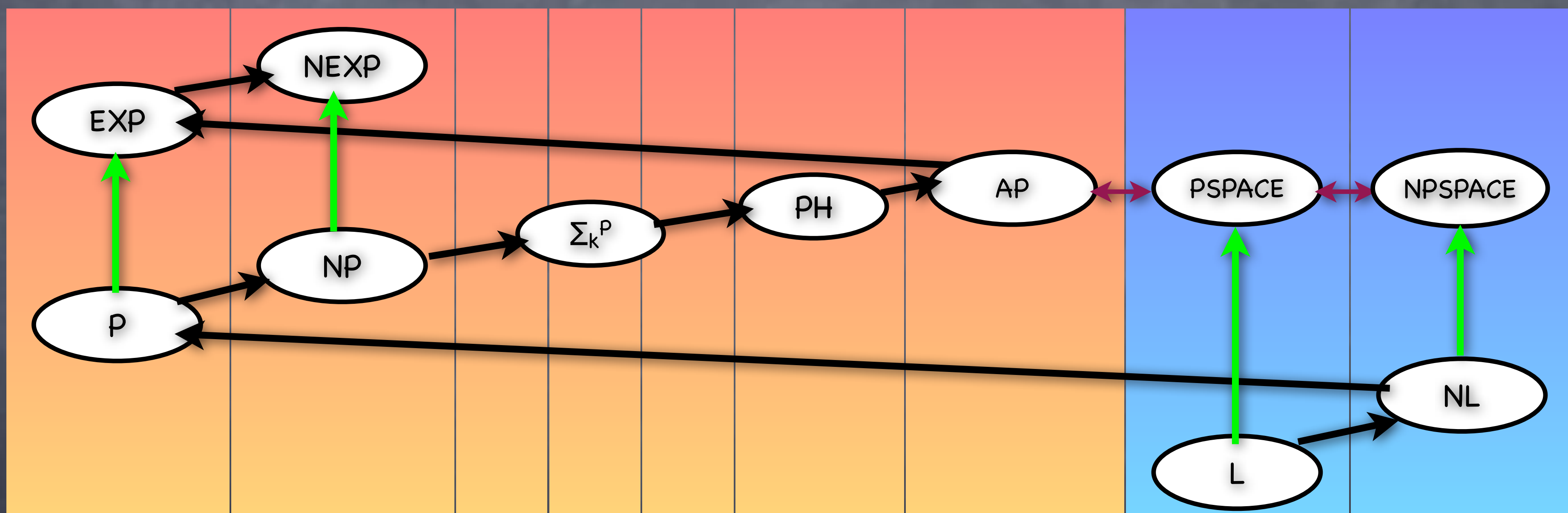
ASPACE vs. DTIME

- ASPACE(S) = DTIME($2^{O(S)}$)
  - APSPACE = EXP

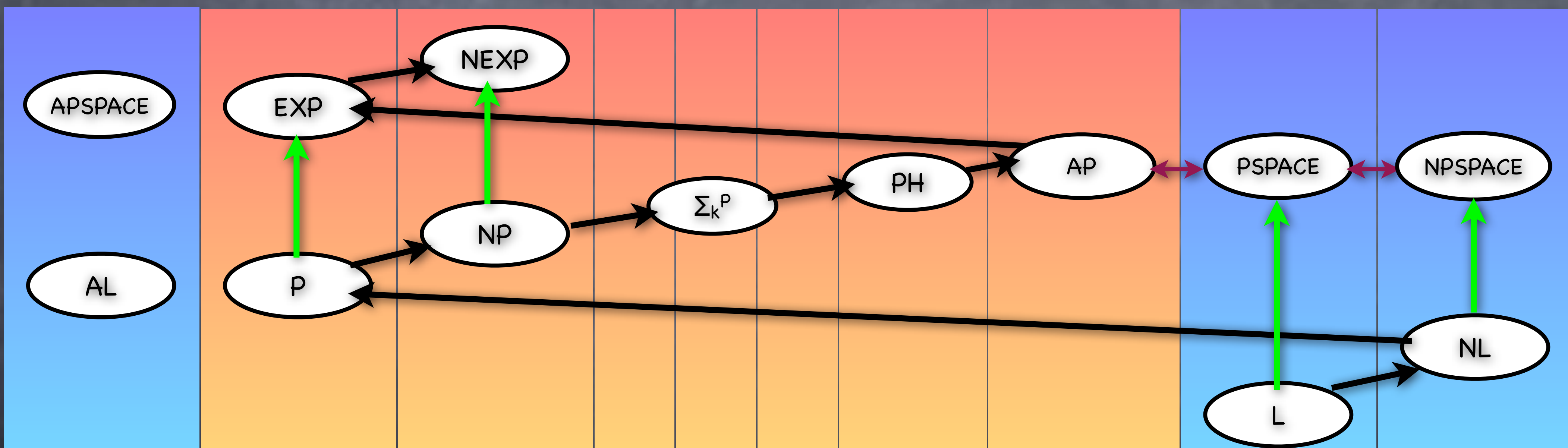
ASPACE vs. DTIME

- ASPACE(S) = DTIME($2^{O(S)}$)
  - APSPACE = EXP
  - AL = P

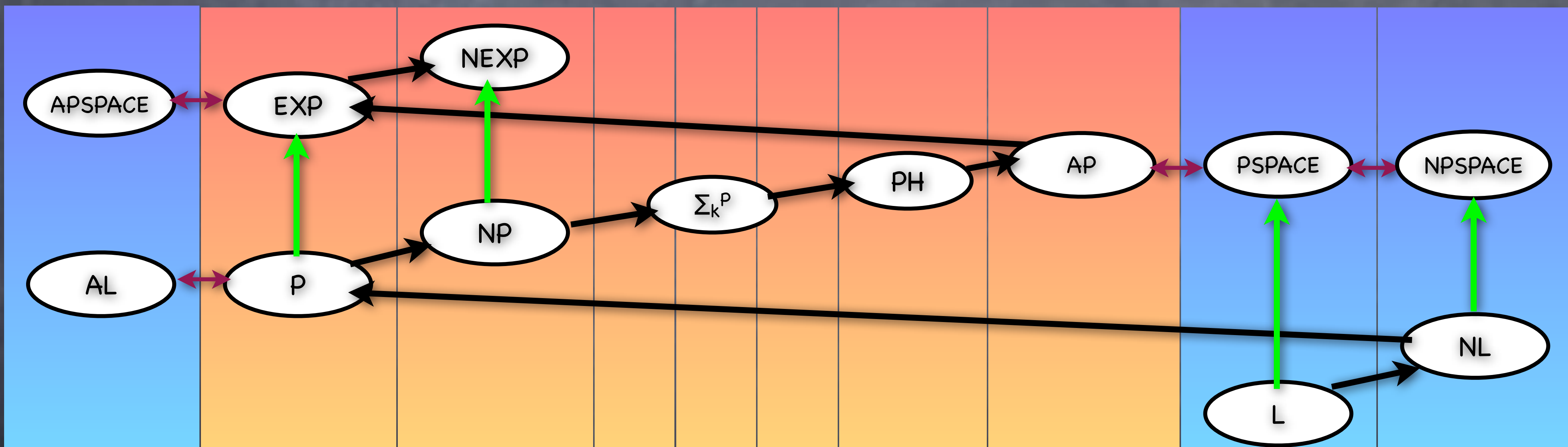
Zoo



Zoo



Zoo



DTISP(T,S)

DTISP(T,S)

- Theorem: NTIME(n) $\not\subset$ DTISP($n^{1+\varepsilon}$,n^{δ}) for some ε, $\delta > 0$

DTISP(T,S)

- Theorem: NTIME(n) $\not\subset$ DTISP($n^{1+\varepsilon}$,n^{δ}) for some ε, $\delta > 0$
- i.e., cannot solve SAT in some slightly super-linear time and slightly super-logarithmic space

DTISP(T,S)

- Theorem: NTIME(n) $\not\subset$ DTISP($n^{1+\varepsilon}$,n^{δ}) for some ε, $\delta > 0$
- i.e., cannot solve SAT in some slightly super-linear time and slightly super-logarithmic space
  - Commonly Believed: can't solve in less than exponential time or with less than linear space

DTISP(T,S)

- Theorem: NTIME(n) $\not\subset$ DTISP($n^{1+\varepsilon}$,n^{δ}) for some ε, $\delta > 0$
- i.e., cannot solve SAT in some slightly super-linear time and slightly super-logarithmic space
  - Commonly Believed: can't solve in less than exponential time or with less than linear space
- Follows (after careful choice of parameters) from

DTISP(T,S)

- Theorem: NTIME(n) $\not\subset$ DTISP($n^{1+\varepsilon}$,n^{δ}) for some ε, $\delta > 0$
- i.e., cannot solve SAT in some slightly super-linear time and slightly super-logarithmic space
  - Commonly Believed: can't solve in less than exponential time or with less than linear space
- Follows (after careful choice of parameters) from
  - DTISP(T,S) $\subseteq$ Σ_2TIME($T^{1/2}$ S)

DTISP(T,S)

- Theorem: $\text{NTIME}(n) \not\subseteq \text{DTISP}(n^{1+\varepsilon}, n^{\delta})$ for some $\varepsilon, \delta > 0$
- i.e., cannot solve SAT in some slightly super-linear time and slightly super-logarithmic space
  - Commonly Believed: can't solve in less than exponential time or with less than linear space
- Follows (after careful choice of parameters) from
  - $\text{DTISP}(T,S) \subseteq \Sigma_2\text{TIME}(T^{1/2}\ S)$

quantification to guess intermediate configs; then check consecutive guesses

DTISP(T,S)

- Theorem: NTIME(n) $\not\subset$ DTISP($n^{1+\varepsilon}$,n^{δ}) for some ε, $\delta > 0$
- i.e., cannot solve SAT in some slightly super-linear time and slightly super-logarithmic space
  - Commonly Believed: can't solve in less than exponential time or with less than linear space
- Follows (after careful choice of parameters) from
  - DTISP(T,S) $\subseteq$ Σ_2TIME($T^{1/2}$ S)
  - NTIME(n) $\subseteq$ DTIME($n^{1+\varepsilon}$) $\Rightarrow$ Σ_2TIME(T) $\subseteq$ NTIME($T^{1+\varepsilon}$)

quantification to guess intermediate configs; then check consecutive guesses

DTISP(T,S)

- Theorem: $\mathrm{NTIME}(n) \not\subseteq \mathrm{DTISP}(n^{1+\varepsilon}, n^{\delta})$ for some $\varepsilon, \delta > 0$
- i.e., cannot solve SAT in some slightly super-linear time and slightly super-logarithmic space
  - Commonly Believed: can't solve in less than exponential time or with less than linear space
- Follows (after careful choice of parameters) from
  - $\mathrm{DTISP}(T,S) \subseteq \Sigma_2\mathrm{TIME}(T^{1/2}\ S)$
  - $\mathrm{NTIME}(n) \subseteq \mathrm{DTIME}(n^{1+\varepsilon}) \Rightarrow \Sigma_2\mathrm{TIME}(T) \subseteq \mathrm{NTIME}(T^{1+\varepsilon})$
    - $\mathrm{NTIME}(n) \subseteq \mathrm{DTISP}(n^{1+\varepsilon}, n^{\delta}) \Rightarrow \mathrm{NTIME}(n^{t}) \subseteq \mathrm{NTIME}(n^{t(1/2+\varepsilon')})$!

quantification to guess intermediate configs; then check consecutive guesses

Today

Today

- ATM to define levels of PH

Today

- ATM to define levels of PH
  - ATIME and ASPACE

Today

- ATM to define levels of PH
  - ATIME and ASPACE
    - AP = PSPACE and APSPACE = EXP

Today

- ATM to define levels of PH
  - ATIME and ASPACE
    - AP = PSPACE and APSPACE = EXP
- Using Σ_2TIME for a DTISP lower-bound