Recall our earlier definitions of space complexities:

**Definition 9.1:** Let \( f : \mathbb{N} \rightarrow \mathbb{R}^+ \) be a function.

1. \( \text{DSpace}(f(n)) \) is the class of all languages \( L \) for which there is an \( O(f(n)) \)-space bounded Turing machine deciding \( L \).
2. \( \text{NSpace}(f(n)) \) is the class of all languages \( L \) for which there is an \( O(f(n)) \)-space bounded nondeterministic Turing machine deciding \( L \).

Being \( O(f(n)) \)-space bounded requires a (nondeterministic) TM

- to halt on every input and
- to use \( \leq f(|w|) \) tape cells on every computation path.
Space Complexity Classes

Some important space complexity classes:

\[
\begin{align*}
L &= \text{LogSpace} = \text{DSpace}(\log n) \\
\text{PSpace} &= \bigcup_{d \geq 1} \text{DSpace}(n^d) \\
\text{ExpSpace} &= \bigcup_{d \geq 1} \text{DSpace}(2^{n^d}) \\
\end{align*}
\]

- \text{LogSpace} = \text{DSpace}(\log n) \quad \text{logarithmic space}
- \text{PSpace} = \bigcup_{d \geq 1} \text{DSpace}(n^d) \quad \text{polynomial space}
- \text{ExpSpace} = \bigcup_{d \geq 1} \text{DSpace}(2^{n^d}) \quad \text{exponential space}

\[
\begin{align*}
\text{NLogSpace} &= \text{NSpace}(\log n) \\
\text{NPSpace} &= \bigcup_{d \geq 1} \text{NSpace}(n^d) \\
\text{NExpSpace} &= \bigcup_{d \geq 1} \text{NSpace}(2^{n^d}) \\
\end{align*}
\]

- \text{NLogSpace} = \text{NSpace}(\log n) \quad \text{nondet. logarithmic space}
- \text{NPSpace} = \bigcup_{d \geq 1} \text{NSpace}(n^d) \quad \text{nondet. polynomial space}
- \text{NExpSpace} = \bigcup_{d \geq 1} \text{NSpace}(2^{n^d}) \quad \text{nondet. exponential space}
The Power of Space

Space seems to be more powerful than time because space can be reused.

Example 9.2: S can be solved in linear space: Just iterate over all possible truth assignments (each linear in size) and check if one satisfies the formula.

Example 9.3: Tautology can be solved in linear space: Just iterate over all possible truth assignments (each linear in size) and check if all satisfy the formula.

More generally: \( \text{NP} \subseteq \text{PSpace} \) and \( \text{coNP} \subseteq \text{PSpace} \)
The Power of Space

Space seems to be more powerful than time because space can be reused.

**Example 9.2:** \textsc{Sat} can be solved in linear space:
Just iterate over all possible truth assignments (each linear in size) and check if one satisfies the formula.
The Power of Space

Space seems to be more powerful than time because space can be reused.

**Example 9.2: SAT** can be solved in linear space:
Just iterate over all possible truth assignments (each linear in size) and check if one satisfies the formula.

**Example 9.3: Tautology** can be solved in linear space:
Just iterate over all possible truth assignments (each linear in size) and check if all satisfy the formula.
The Power of Space

Space seems to be more powerful than time because space can be reused.

**Example 9.2:** SAT can be solved in linear space:
Just iterate over all possible truth assignments (each linear in size) and check if one satisfies the formula.

**Example 9.3:** Tautology can be solved in linear space:
Just iterate over all possible truth assignments (each linear in size) and check if all satisfy the formula.

More generally: \( \text{NP} \subseteq \text{PSpace} \) and \( \text{coNP} \subseteq \text{PSpace} \)
**Theorem 9.4:** For every function $f : \mathbb{N} \rightarrow \mathbb{R}^+$, for all $c \in \mathbb{N}$, and for every $f$-space bounded (deterministic/nondeterministic) Turing machine $M$:

there is a $\max\{1, \frac{1}{c} f(n)\}$-space bounded (deterministic/nondeterministic) Turing machine $M'$ that accepts the same language as $M$. 

Proof idea: Similar to (but much simpler than) linear speed-up. □

This justifies using $O$-notation for defining space classes.
Theorem 9.4: For every function \( f : \mathbb{N} \rightarrow \mathbb{R}^+ \), for all \( c \in \mathbb{N} \), and for every \( f \)-space bounded (deterministic/nondeterministic) Turing machine \( M \): there is a \( \max\{1, \frac{1}{c} f(n)\} \)-space bounded (deterministic/nondeterministic) Turing machine \( M' \) that accepts the same language as \( M \).

Proof idea: Similar to (but much simpler than) linear speed-up.
Theorem 9.4: For every function $f : \mathbb{N} \to \mathbb{R}^+$, for all $c \in \mathbb{N}$, and for every $f$-space bounded (deterministic/nondeterministic) Turing machine $M$: there is a $\max\{1, \frac{1}{c}f(n)\}$-space bounded (deterministic/nondeterministic) Turing machine $M'$ that accepts the same language as $M$.

Proof idea: Similar to (but much simpler than) linear speed-up. □

This justifies using $O$-notation for defining space classes.
**Theorem 9.5:** For every function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ all $k \geq 1$ and $L \subseteq \Sigma^*$:

If $L$ can be decided by an $f$-space bounded $k$-tape Turing-machine, then it can also be decided by an $f$-space bounded 1-tape Turing-machine.

\[ \square \]

Note: We still use a separate read-only input tape to define some space complexities, such as LogSpace.
Theorem 9.5: For every function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ all $k \geq 1$ and $L \subseteq \Sigma^*$:

If $L$ can be decided by an $f$-space bounded $k$-tape Turing-machine, then it can also be decided by an $f$-space bounded 1-tape Turing-machine.

Proof idea: Combine tapes with a similar reduction as for time. Compress space to avoid linear increase.

Note: We still use a separate read-only input tape to define some space complexities, such as LogSpace.
Theorem 9.6: For all functions $f : \mathbb{N} \to \mathbb{R}^+$:

$$\text{DTime}(f) \subseteq \text{DSpace}(f) \quad \text{and} \quad \text{NTime}(f) \subseteq \text{NSpace}(f)$$

Proof: Visiting a cell takes at least one time step. □
Theorem 9.6: For all functions $f : \mathbb{N} \to \mathbb{R}^+$:

$$
\text{DTime}(f) \subseteq \text{DSpace}(f) \quad \text{and} \quad \text{NTime}(f) \subseteq \text{NSpace}(f)
$$

Proof: Visiting a cell takes at least one time step. \hfill \Box

Theorem 9.7: For all functions $f : \mathbb{N} \to \mathbb{R}^+$ with $f(n) \geq \log n$:

$$
\text{DSpace}(f) \subseteq \text{DTime}(2^{O(f)}) \quad \text{and} \quad \text{NSpace}(f) \subseteq \text{DTime}(2^{O(f)})
$$

Markus Krötzsch, 15th Nov 2017
**Theorem 9.6:** For all functions $f : \mathbb{N} \to \mathbb{R}^+$:

\[ \text{DTime}(f) \subseteq \text{DSpace}(f) \quad \text{and} \quad \text{NTime}(f) \subseteq \text{NSpace}(f) \]

**Proof:** Visiting a cell takes at least one time step. \qed

**Theorem 9.7:** For all functions $f : \mathbb{N} \to \mathbb{R}^+$ with $f(n) \geq \log n$:

\[ \text{DSpace}(f) \subseteq \text{DTime}(2^{O(f)}) \quad \text{and} \quad \text{NSpace}(f) \subseteq \text{DTime}(2^{O(f)}) \]

**Proof:** Based on configuration graphs and a bound on the number of possible configurations.
Number of Possible Configurations

Let $\mathcal{M} := (Q, \Sigma, \Gamma, q_0, \delta, q_{\text{start}})$ be a 2-tape Turing machine
(1 read-only input tape + 1 work tape)

Recall: A configuration of $\mathcal{M}$ is a quadruple $(q, p_1, p_2, x)$ where

- $q \in Q$ is the current state,
- $p_i \in \mathbb{N}$ is the head position on tape $i$, and
- $x \in \Gamma^*$ is the tape content.

Let $w \in \Sigma^*$ be an input to $\mathcal{M}$ and $n := |w|$.

- Then also $p_1 \leq n$.
- If $\mathcal{M}$ is $f(n)$-space bounded we can assume $p_2 \leq f(n)$ and $|x| \leq f(n)$
Number of Possible Configurations

Let $\mathcal{M} := (Q, \Sigma, \Gamma, q_0, \delta, q_{\text{start}})$ be a 2-tape Turing machine (1 read-only input tape + 1 work tape)

Recall: A configuration of $\mathcal{M}$ is a quadruple $(q, p_1, p_2, x)$ where

- $q \in Q$ is the current state,
- $p_i \in \mathbb{N}$ is the head position on tape $i$, and
- $x \in \Gamma^*$ is the tape content.

Let $w \in \Sigma^*$ be an input to $\mathcal{M}$ and $n := |w|$.

- Then also $p_1 \leq n$.
- If $\mathcal{M}$ is $f(n)$-space bounded we can assume $p_2 \leq f(n)$ and $|x| \leq f(n)$

Hence, there are at most

$$|Q| \cdot n \cdot f(n) \cdot |\Gamma|^{f(n)} = n \cdot 2^{O(f(n))} = 2^{O(f(n))}$$

different configurations on inputs of length $n$ (the last equality requires $f(n) \geq \log n$).
Configuration Graphs

The possible computations of a TM $M$ (on input $w$) form a directed graph:

- Vertices: configurations that $M$ can reach (on input $w$)
- Edges: there is an edge from $C_1$ to $C_2$ if $C_1 \vdash_M C_2$ ($C_2$ reachable from $C_1$ in a single step)

This yields the configuration graph:

- Could be infinite in general.
- For $f(n)$-space bounded 2-tape TMs, there can be at most $2^{O(f(n))}$ vertices and $2 \cdot (2^{O(f(n))})^2 = 2^{O(f(n))}$ edges
Configuration Graphs

The possible computations of a TM $M$ (on input $w$) form a directed graph:

- Vertices: configurations that $M$ can reach (on input $w$)
- Edges: there is an edge from $C_1$ to $C_2$ if $C_1 \vdash_M C_2$
  ($C_2$ reachable from $C_1$ in a single step)

This yields the configuration graph:

- Could be infinite in general.
- For $f(n)$-space bounded 2-tape TMs,
  there can be at most $2^{O(f(n))}$ vertices and $2 \cdot (2^{O(f(n))})^2 = 2^{O(f(n))}$ edges

A computation of $M$ on input $w$ corresponds to a path in the configuration graph from the start configuration to a stop configuration.

Hence, to test if $M$ accepts input $w$,

- construct the configuration graph and
- find a path from the start to an accepting stop configuration.
**Theorem 9.6:** For all functions \( f : \mathbb{N} \to \mathbb{R}^+ \):

\[
\text{DTime}(f) \subseteq \text{DSpace}(f) \quad \text{and} \quad \text{NTime}(f) \subseteq \text{NSpace}(f)
\]

**Proof:** Visiting a cell takes at least one time step. \( \square \)

**Theorem 9.7:** For all functions \( f : \mathbb{N} \to \mathbb{R}^+ \) with \( f(n) \geq \log n \):

\[
\text{DSpace}(f) \subseteq \text{DTime}(2^{O(f)}) \quad \text{and} \quad \text{NSpace}(f) \subseteq \text{DTime}(2^{O(f)})
\]

**Proof:** Build the configuration graph (time \( 2^{O(f(n))} \)) and find a path from the start to an accepting stop configuration (time \( 2^{O(f(n))} \)). \( \square \)
Applying the results of the previous slides, we get the following relations:

\[ L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace \subseteq NPSpace \subseteq ExpTime \subseteq NExpTime \]

We also noted \( P \subseteq coNP \subseteq PSpace \).

Open questions:

- What is the relationship between space classes and their co-classes?
- What is the relationship between deterministic and non-deterministic space classes?
Most experts think that nondeterministic TMs can solve strictly more problems when given the same amount of time than a deterministic TM:

Most believe that $P \subseteq NP$

How about nondeterminism in space-bounded TMs?
Most experts think that nondeterministic TMs can solve strictly more problems when given the same amount of time than a deterministic TM:

Most believe that $P \subseteq NP$

How about nondeterminism in space-bounded TMs?

**Theorem 9.8 (Savitch’s Theorem, 1970):** For any function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ with $f(n) \geq \log n$:

$$\text{NSpace}(f(n)) \subseteq \text{DSpace}(f^2(n)).$$

That is: nondeterminism adds **almost** no power to space-bounded TMs!
Consequences of Savitch’s Theorem

**Theorem 9.8 (Savitch’s Theorem, 1970):** For any function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ with $f(n) \geq \log n$:

$$\text{NSpace}(f(n)) \subseteq \text{DSpace}(f^2(n)).$$

**Corollary 9.9:** $\text{PSPACE} = \text{NPSPACE}$.

**Proof:** $\text{PSPACE} \subseteq \text{NPSPACE}$ is clear. The converse follows since the square of a polynomial is still a polynomial. $\square$

Similarly for “bigger” classes, e.g., $\text{EXPSPACE} = \text{NEXPSPACE}$.

**Corollary 9.10:** $\text{NL} \subseteq \text{DSpace}(O(\log_2 n))$.

Note that $\log_2(n) < O(\log n)$, so we do not obtain $\text{NL} = \text{L}$ from this.
Theorem 9.8 (Savitch’s Theorem, 1970): For any function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ with $f(n) \geq \log n$:

$$\text{NSpace}(f(n)) \subseteq \text{DSpace}(f^2(n)).$$

Corollary 9.9: $\text{PSPACE} = \text{NPSPACE}$.

Proof: $\text{PSPACE} \subseteq \text{NPSPACE}$ is clear. The converse follows since the square of a polynomial is still a polynomial.

Similarly for “bigger” classes, e.g., $\text{EXPSPACE} = \text{NEXPSPACE}$.
Theorem 9.8 (Savitch’s Theorem, 1970): For any function \( f : \mathbb{N} \rightarrow \mathbb{R}^+ \) with \( f(n) \geq \log n \):

\[
\text{NSpace}(f(n)) \subseteq \text{DSpace}(f^2(n)).
\]

Corollary 9.9: \( \text{PSpace} = \text{NPSpace} \).

Proof: \( \text{PSpace} \subseteq \text{NPSpace} \) is clear. The converse follows since the square of a polynomial is still a polynomial. \[\Box\]

Similarly for “bigger” classes, e.g., \( \text{ExpSpace} = \text{NExpSpace} \).

Corollary 9.10: \( \text{NL} \subseteq \text{DSpace}(O(\log^2 n)) \).

Note that \( \log^2(n) \notin O(\log n) \), so we do not obtain \( \text{NL} = \text{L} \) from this.
Simulating nondeterminism with more space:

- Use configuration graph of nondeterministic space-bounded TM
- Check if an accepting configuration can be reached
- Store only one computation path at a time (depth-first search)

This still requires exponential space. We want quadratic space!

What to do?

Things we can do:

- Store one configuration:
  - one configuration requires \( \log n + O(f(n)) \) space
  - if \( f(n) \geq \log n \), then this is \( O(f(n)) \) space

- Store \( \log n \) configurations (remember we have \( \log 2n \) space)

- Iterate over all configurations (one by one)
Proving Savitch’s Theorem

Simulating nondeterminism with more space:
- Use configuration graph of nondeterministic space-bounded TM
- Check if an accepting configuration can be reached
- Store only one computation path at a time (depth-first search)

This still requires exponential space. We want quadratic space!

What to do?
Simulating nondeterminism with more space:

- Use configuration graph of nondeterministic space-bounded TM
- Check if an accepting configuration can be reached
- Store only one computation path at a time (depth-first search)

This still requires exponential space. We want quadratic space!

**What to do?**

**Things we can do:**

- Store one configuration:
  - one configuration requires $\log n + O(f(n))$ space
  - if $f(n) \geq \log n$, then this is $O(f(n))$ space
- Store $\log n$ configurations (remember we have $\log^2 n$ space)
- Iterate over all configurations (one by one)
To find out if we can reach an accepting configuration, we solve a slightly more general question:

**Yieldability**

Input: TM configurations $C_1$ and $C_2$, integer $k$

Problem: Can TM get from $C_1$ to $C_2$ in at most $k$ steps?
To find out if we can reach an accepting configuration, we solve a slightly more general question:

**Yieldability**

Input: TM configurations $C_1$ and $C_2$, integer $k$

Problem: Can TM get from $C_1$ to $C_2$ in at most $k$ steps?

**Approach:** check if there is an intermediate configuration $C'$ such that

1. $C_1$ can reach $C'$ in $k/2$ steps and
2. $C'$ can reach $C_2$ in $k/2$ steps

$\rightarrow$ **Deterministic:** we can try all $C'$ (iteration)

$\rightarrow$ **Space-efficient:** we can reuse the same space for both steps
An Algorithm for Yieldability

```plaintext
01 CanYield(C1, C2, k) {
02     if k = 1 :
03         return (C1 = C2) or (C1 ⊢_M C2)
04     else if k > 1 :
05         for each configuration C of M for input size n :
06             if CanYield(C1, C, k/2) and
07                 CanYield(C, C2, k/2) :
08                 return true
09         // eventually, if no success:
10         return false
11 }
```

- We only call CanYield only with $k$ a power of 2, so $k/2 \in \mathbb{N}$
CanYield$(C_1, C_2, k)$ {
  if $k = 1$ :
    return $(C_1 = C_2)$ or $(C_1 \vdash_M C_2)$
  else if $k > 1$ :
    for each configuration $C$ of $M$ for input size $n$ :
      if CanYield$(C_1, C, k/2)$ and
      CanYield$(C, C_2, k/2)$ :
        return true
    // eventually, if no success:
    return false
}

During iteration (line 5), we store one $C$ in $O(f(n))$.

Calls in lines 6 and 7 can reuse the same space.

Maximum depth of recursive call stack: $\log_2 k$.

Overall space usage: $O(f(n) \cdot \log k)$.
Space Requirement for the Algorithm

```plaintext
01 CanYield(C_1, C_2, k) {
02     if k = 1 :
03         return (C_1 = C_2) or (C_1 ⊢_M C_2)
04     else if k > 1 :
05         for each configuration C of M for input size n :
06             if CanYield(C_1, C, k/2) and
07                 CanYield(C, C_2, k/2) :
08                 return true
09         // eventually, if no success:
10     return false
11 }
```

- During iteration (line 05), we store one $C$ in $O(f(n))$
Space Requirement for the Algorithm

```c
01 CanYield(C1,C2,k) {
02     if k == 1 :
03         return (C1 == C2) or (C1 ⊨M C2)
04     else if k > 1 :
05         for each configuration C of M for input size n :
06             if CanYield(C1,C,k/2) and 
07                 CanYield(C,C2,k/2) :
08                 return true
09         // eventually, if no success:
10         return false
11 }
```

- During iteration (line 05), we store one $C$ in $O(f(n))$
- Calls in lines 06 and 07 can reuse the same space
Space Requirement for the Algorithm

```c
01 CanYield(C_1, C_2, k) {
02   if k = 1 :
03       return (C_1 = C_2) or (C_1 ⊬_M C_2)
04   else if k > 1 :
05       for each configuration C of M for input size n :
06           if CanYield(C_1, C, k/2) and
07               CanYield(C, C_2, k/2) :
08               return true
09         // eventually, if no success:
10       return false
11 }
```

- During iteration (line 05), we store one $C$ in $O(f(n))$
- Calls in lines 06 and 07 can reuse the same space
- Maximum depth of recursive call stack: $\log_2 k$
Space Requirement for the Algorithm

```
01 CanYield(C1, C2, k) {
02   if k = 1 :
03     return (C1 = C2) or (C1 ⊢M C2)
04   else if k > 1 :
05     for each configuration C of M for input size n :
06       if CanYield(C1, C, k/2) and
07         CanYield(C, C2, k/2) :
08         return true
09     // eventually, if no success:
10   return false
11 }
```

- During iteration (line 05), we store one \( C \) in \( O(f(n)) \)
- Calls in lines 06 and 07 can reuse the same space
- Maximum depth of recursive call stack: \( \log_2 k \)

Overall space usage: \( O(f(n) \cdot \log k) \)
Simulating Nondeterministic Space-Bounded TMs

**Input:** TM $M$ that runs in $\text{NSpace}(f(n))$; input word $w$ of length $n$

**Algorithm:**
- Modify $M$ to have a unique accepting configuration $C_{\text{accept}}$: when accepting, erase tape and move head to the very left
- Select $d$ such that $2^{df(n)} \geq |Q| \cdot n \cdot f(n) \cdot |\Gamma|^f(n)$
- Return $\text{CanYield}(C_{\text{start}}, C_{\text{accept}}, k)$ with $k = 2^{df(n)}$
Simulating Nondeterministic Space-Bounded TMs

**Input:** TM $M$ that runs in $\text{NSpace}(f(n))$; input word $w$ of length $n$

**Algorithm:**

- Modify $M$ to have a unique accepting configuration $C_{\text{accept}}$:
  when accepting, erase tape and move head to the very left
- Select $d$ such that $2^{df(n)} \geq |Q| \cdot n \cdot f(n) \cdot |\Gamma|^{f(n)}$
- Return $\text{CanYield}(C_{\text{start}}, C_{\text{accept}}, k)$ with $k = 2^{df(n)}$

**Space requirements:**

$\text{CanYield}$ runs in space

$$O(f(n) \cdot \log k) = O\left(f(n) \cdot \log 2^{df(n)}\right) = O(f(n) \cdot df(n)) = O(f^2(n))$$
How does the algorithm actually do this?

• $f(n)$ was not part of the input!
• Even if we knew $f$, it might not be easy to compute!

Solution: replace $f(n)$ by a parameter $\ell$ and probe its value

1. Start with $\ell = 1$
2. Check if $M$ can reach any configuration with more than $\ell$ tape cells (iterate over all configurations of size $\ell + 1$; use CanYield on each)
3. If yes, increase $\ell$ by 1; goto (2)
4. Run algorithm as before, with $f(n)$ replaced by $\ell$

Therefore: we don't need to know $f$ at all. This finishes the proof. $\blacksquare$
“Select $d$ such that $2^{df(n)} \geq |Q| \cdot n \cdot f(n) \cdot |\Gamma|^f(n)$”

How does the algorithm actually do this?
“Select $d$ such that $2^{df(n)} \geq |Q| \cdot n \cdot f(n) \cdot |\Gamma|^f(n)$”

How does the algorithm actually do this?

- $f(n)$ was not part of the input!
- Even if we knew $f$, it might not be easy to compute!
Did We Really Do It?

“Select $d$ such that $2^{df(n)} \geq |Q| \cdot n \cdot f(n) \cdot |\Gamma|^f(n)$”

How does the algorithm actually do this?

- $f(n)$ was not part of the input!
- Even if we knew $f$, it might not be easy to compute!

Solution: replace $f(n)$ by a parameter $\ell$ and probe its value

1. Start with $\ell = 1$
2. Check if $M$ can reach any configuration with more than $\ell$ tape cells
   (iterate over all configurations of size $\ell + 1$; use CanYield on each)
3. If yes, increase $\ell$ by 1; goto (2)
4. Run algorithm as before, with $f(n)$ replaced by $\ell$

Therefore: we don’t need to know $f$ at all. This finishes the proof. $\square$
Summary: Relationships of Space and Time

Summing up, we get the following relations:

\[
L \subseteq NL \subseteq P \subseteq NP \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXPTIME} \subseteq \text{NEXPSPACE}
\]

We also noted \( P \subseteq \text{coNP} \subseteq \text{PSPACE} \).

**Open questions:**

- Is Savitch’s Theorem tight?
- Are there any interesting problems in these space classes?
- We have \( \text{PSPACE} = \text{NPSPACE} = \text{coNPSPACE} \).
  But what about \( L, NL, \) and \( \text{coNL} \)?
Summing up, we get the following relations:

\[ L \subseteq NL \subseteq P \subseteq NP \subseteq PSpace = NPSpace \subseteq ExpTime \subseteq NExpTime \]

We also noted \( P \subseteq coNP \subseteq PSpace \).

**Open questions:**

- Is Savitch’s Theorem tight?
- Are there any interesting problems in these space classes?
- We have \( PSpace = NPSpace = coNPSpace \).
  But what about \( L \), \( NL \), and \( coNL \)?

\→ the first: nobody knows (YCTBF); the others: see upcoming lectures