Computing With A Full Hard Drive

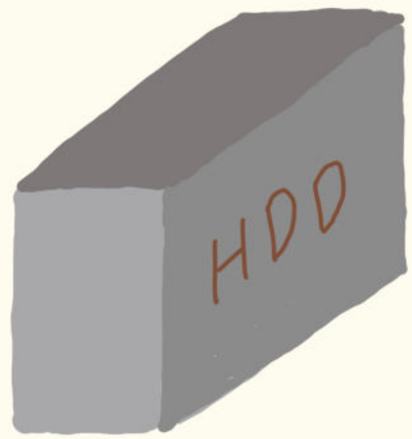
Ian Mertz
Charles University

Theory of Computation

Time

runtime

Space



memory

THEORY OF COMPUTATION

TIME

- speed of computation (e.g. # of processor cycles)

SPACE

THEORY OF COMPUTATION

TIME

- speed of computation (e.g. # of processor cycles)
- most common resource to optimize in practice

SPACE

THEORY OF COMPUTATION

TIME

- speed of computation (e.g. # of processor cycles)
- most common resource to optimize in practice
- Moore's Law (processors)

SPACE

THEORY OF COMPUTATION

TIME

- speed of computation (e.g. # of processor cycles)
- most common resource to optimize in practice
- Moore's Law (processors)
- big questions: P vs NP, power of randomness, etc.

SPACE

Theory of Computation

Time

- speed of computation (e.g. # of processor cycles)
- most common resource to optimize in practice
- Moore's Law (processors)
- big questions: P vs NP, power of randomness, etc.

Space

- computing under memory constraints

Theory of Computation

Time

- speed of computation (e.g. # of processor cycles)
- most common resource to optimize in practice
- Moore's Law (processors)
- big questions: P vs NP, power of randomness, etc.

Space

- computing under memory constraints
- bottleneck for distributed systems, quantum computing

Theory of Computation

Time	Space
- speed of computation (e.g. # of processor cycles)	- computing under memory constraints
- most common resource to optimize in practice	- bottleneck for distributed systems, quantum computing
- Moore's Law (processors)	- Moore's Law (chips)
- big questions: P vs NP, power of randomness, etc.	

Theory of Computation

Time	Space
- speed of computation (e.g. # of processor cycles)	- computing under memory constraints
- most common resource to optimize in practice	- bottleneck for distributed systems, quantum computing
- Moore's Law (processors)	- Moore's Law (chips)
- big questions: P vs NP, power of randomness, etc.	- same questions as time... but very different answers!

Theory of Computation

Time

- speed of computation

ONCE IT'S USED, GONE FOREVER

power of randomness, etc.

Space

- computing under memory constraints
- bottleneck for distributed systems, quantum computing
- Moore's Law (chips)
- same questions as time... but very different answers!

Theory of Computation

Time

- speed of computation

( )

ONCE IT'S

USED, GONE

FOREVER

o

s)

power of randomness, etc.

Space

- computing under memory

ted

ting

CAN BE

ERASED

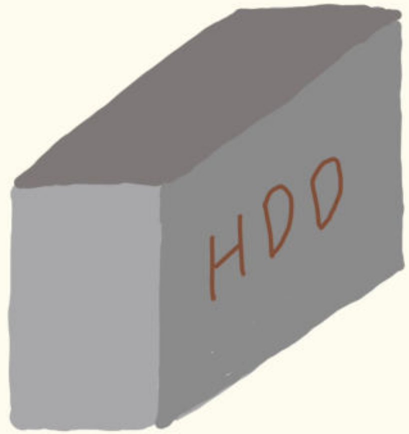
AND REUSED

but very different answers!

Theory of Computation

Time

Space



Fundamental Question

What is the relationship between TIME and SPACE?

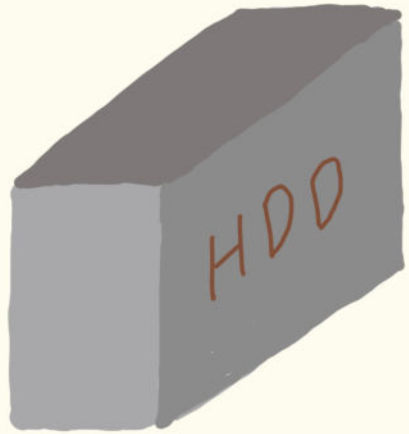
Theory of Computation

Time

TIME[t]

every function computable in at most t steps

Space



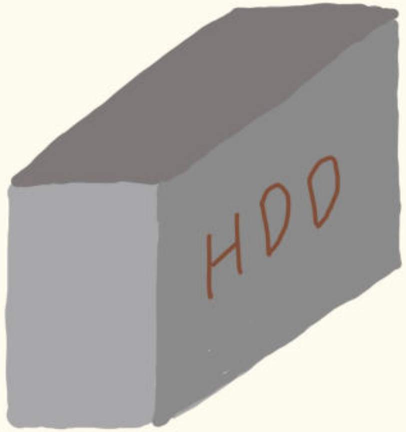
SPACE[s]

every function computable with at most t memory

THEORY OF COMPUTATION

TIME

SPACE



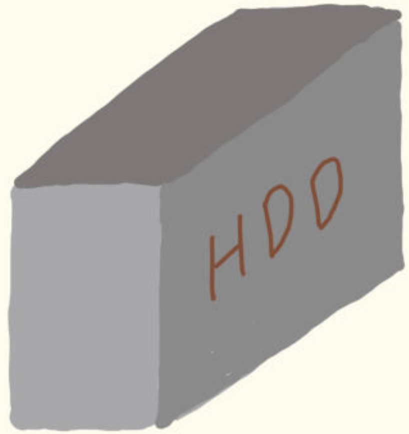
$$\mathrm{TIME}[t] \subseteq \mathrm{SPACE}[t]$$

can only touch
1 bit per step

Theory of Computation

Time

Space



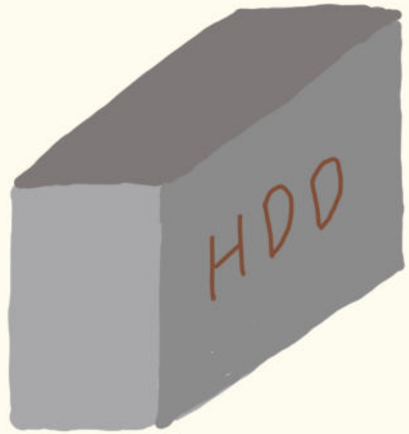
$\mathrm{SPACE}[t] \subseteq \mathrm{TIME}[2^t]$

at most 2^t different settings of t bits

Theory of Computation

Time

Space

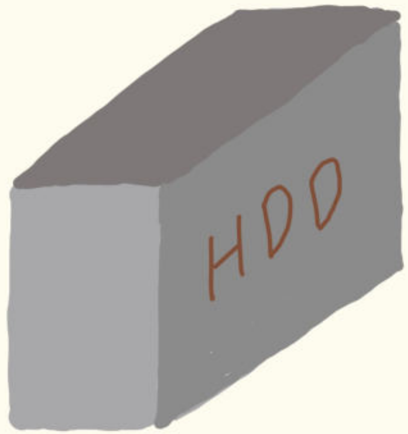


$$\mathrm{TIME}[t] \subseteq \mathrm{SPACE}[t] \subseteq \mathrm{TIME}[2^t]$$

Theory of Computation

Time

Space

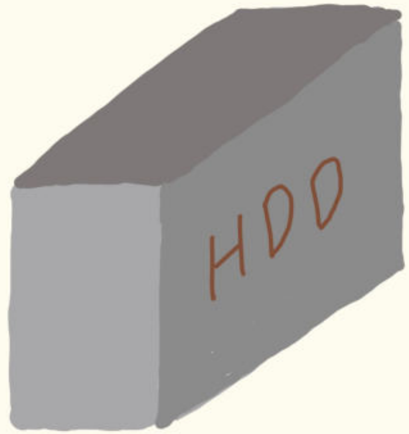


$$\mathrm{TIME}[t] \neq \mathrm{TIME}[2^t]$$

more time = more functions

Theory of Computation

Time | **Space**



$$\mathrm{TIME}[t] \subseteq \mathrm{SPACE}[t] \subseteq \mathrm{TIME}[2^t]$$

one of these is <u>not</u> an equality!

(...but we don't know which)

Theory of Computation

$$\mathrm{TIME}[t] \subseteq \mathrm{SPACE}[t] \subseteq \mathrm{TIME}[2^t]$$

f in $\mathrm{SPACE}[t]$ but <u>not</u> $\mathrm{TIME}[t]$

AND/OR

f in $\mathrm{TIME}[2^t]$ but <u>not</u> $\mathrm{SPACE}[t]$

Theory of Computation

Algorithms

"upper bounds"

Complexity

"lower bounds"

Theory of Computation

Algorithms

"upper bounds"

- study of what is possible

Complexity

"lower bounds"

Theory of Computation

Algorithms

"upper bounds"

- study of what is possible
- chiefly limited by human ingenuity (or lack thereof)

Complexity

"lower bounds"

THEORY OF COMPUTATION

ALGORITHMS

"upper bounds"

- study of what is possible
- chiefly limited by human ingenuity (or lack thereof)
- can translate to practical implementations

COMPLEXITY

"lower bounds"

Theory of Computation

Algorithms

"upper bounds"

- study of what is possible
- chiefly limited by human ingenuity (or lack thereof)
- can translate to practical implementations

Complexity

"lower bounds"

- study of what is impossible

THEORY OF COMPUTATION

ALGORITHMS

"upper bounds"

- study of what is possible
- chiefly limited by human ingenuity (or lack thereof)
- can translate to practical implementations

COMPLEXITY

"lower bounds"

- study of what is impossible
- need to disprove <u>every</u> algorithm (discovered or not)

Theory of Computation

Algorithms

"upper bounds"

- study of what is possible
- chiefly limited by human ingenuity (or lack thereof)
- can translate to practical implementations

Complexity

"lower bounds"

- study of what is impossible
- need to disprove <u>every</u> algorithm (discovered or not)
- tells us when to look for a different approach

THEORY OF COMPUTATION

ALGORITHMS

"upper bounds"

- study of what is possible
- [illegible]

DIFFICULT

- can translate to practical implementations

COMPLEXITY

"lower bounds"

- study of what is impossible
- need to disprove <u>every</u> algorithm (discovered or not)
- tells us when to look for a different approach

THEORY OF COMPUTATION

ALGORITHMS

"upper bounds"

- study of what is possible

DIFFICULT

- can translate to practical implementations

COMPLEXITY

"lower bounds"

-s EXTREMELY

-r SUPER-DUPER

a RIDICULOUSLY t)

-t DIFFICULT for

a

Theory of Computation

Example: TIME vs SPACE

$$\text{TIME}[t] \subseteq \text{SPACE}[t] \subseteq \text{TIME}[2^t]$$

f in SPACE[t] but not TIME[t] AND/OR f in TIME[2^t] but not SPACE[t]

Theory of Computation

Example : TIME vs SPACE

$$\mathrm{SPACE}[t] \subseteq \mathrm{TIME}[2^t]$$

Goal: $\boxed{\begin{array}{c} f \text{ in } \mathrm{TIME}[2^t] \\ \text{but not } \mathrm{SPACE}[t] \end{array}}$

THEORY OF COMPUTATION

Example : TIME vs SPACE

Goal : $\mathrm{TIME}[2^t]$ requires $\mathrm{SPACE}[\gg t]$

THEORY OF COMPUTATION

Example: TIME vs SPACE

Goal: $\text{TIME}[2^t]$ requires $\text{SPACE}[\gg t]$

[CMWBS'12]: $\boxed{f^* \text{ in } \text{TIME}[2^t] \text{ but not } \text{SPACE}[<t^2]}$

"obvious" algorithm: $\text{TIME}[2^t]$
$\text{SPACE}[t^2]$

Theory of Computation

Example: TIME vs SPACE

Goal: TIME[2^t] requires SPACE[$\gg t$]

[CMWBS'12]: $\boxed{f^* \text{ in TIME}[2^t] \text{ but not SPACE}[< t^2]}$

Need to show:

"obvious" algorithm
for f^* is space-optimal

Theory of Computation

Example: TIME vs SPACE

Goal: TIME[2^t] requires SPACE[$\gg t$]

[CMWBS'12]: f^* in TIME[2^t] but not SPACE[$<t^2$]

Need to show:

"obvious" algorithm for f^* is space-optimal	"trivial" facts

Theory of Computation

Example: TIME vs SPACE

Goal: TIME[2^t] requires SPACE[$\gg t$]

[CMWBS'12]: f^* in TIME[2^t] but not SPACE[$< t^2$]

Need to show: "obvious" algorithm for f^* is space-optimal | "trivial" facts

Toy cases: . . . . .

THEORY OF COMPUTATION

Example : TIME vs SPACE

Goal: TIME[2^t] requires SPACE[$\gg t$]

[CMWBS'12]: $\boxed{f^* \text{ in TIME}[2^t] \text{ but not SPACE}[< t^2]}$

Need to show:

"obvious" algorithm for f^* is space-optimal

"trivial" facts

Toy cases: ✓ . . . . . . ✓ ✓

[CMWBS'12]
[L'13]
[EMP'18]
[IN'19]

THEORY OF COMPUTATION

Example : TIME vs SPACE

one "trivial" fact :

THEORY OF COMPUTATION

Example: TIME vs SPACE

one "trivial" fact:

1. COMPUTE f  SPACE(f)

2. REMEMBER z 

SPACE(z)

THEORY OF COMPUTATION

Example: TIME vs SPACE

one "trivial" fact:

1. COMPUTE f  SPACE(f)

2. REMEMBER z 

SPACE(z)

3. BOTH

f 

z 

SPACE(f, z)

THEORY OF COMPUTATION

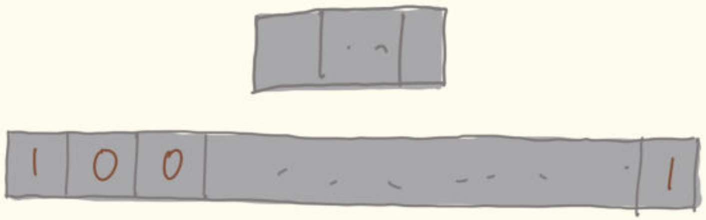
Example : TIME vs SPACE

one "trivial" fact :

1. COMPUTE f

2. REMEMBER z

3. BOTH f z



Q:

SPACE(f)
+
SPACE(z)

equals?

SPACE(f, z)

THEORY OF COMPUTATION

Example : TIME vs SPACE

one "trivial" fact :

Option #1 :

Theory of Computation

Example: TIME vs SPACE

one "trivial" fact:

Option #1:

Option #2:

+

completely full,
must be returned
to its starting data

(z)

Theory of Computation

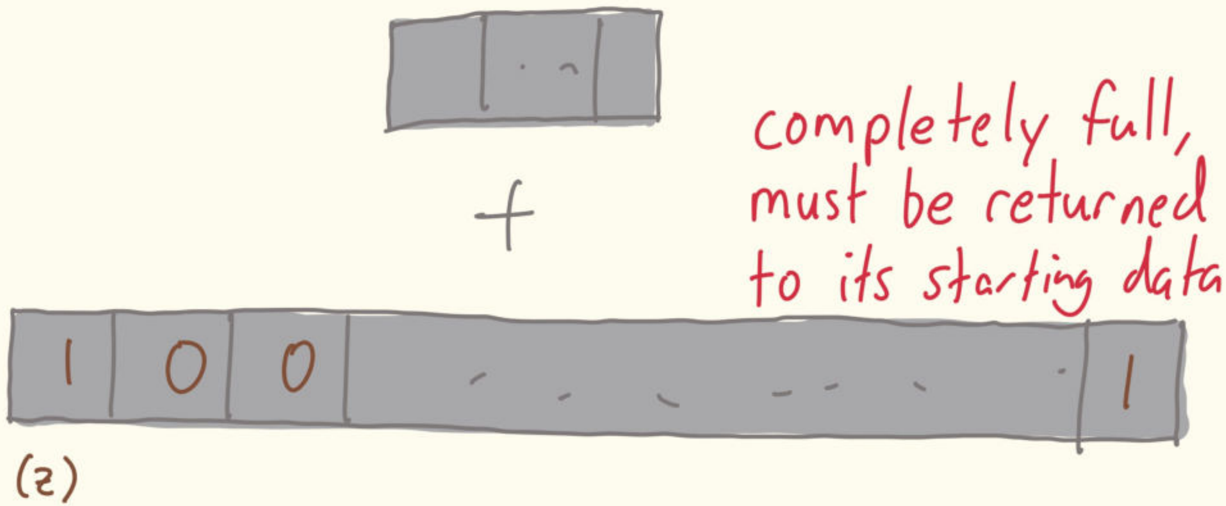
Example : TIME vs SPACE

one "trivial" fact :

Q : is there an f solvable by #2 but not #1 ?

Option #1 :

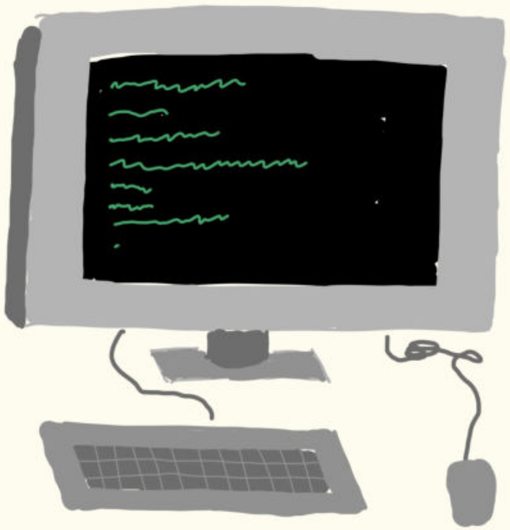
Option #2 :



THEORY OF COMPUTATION

Example : TIME vs SPACE

one "trivial" fact :

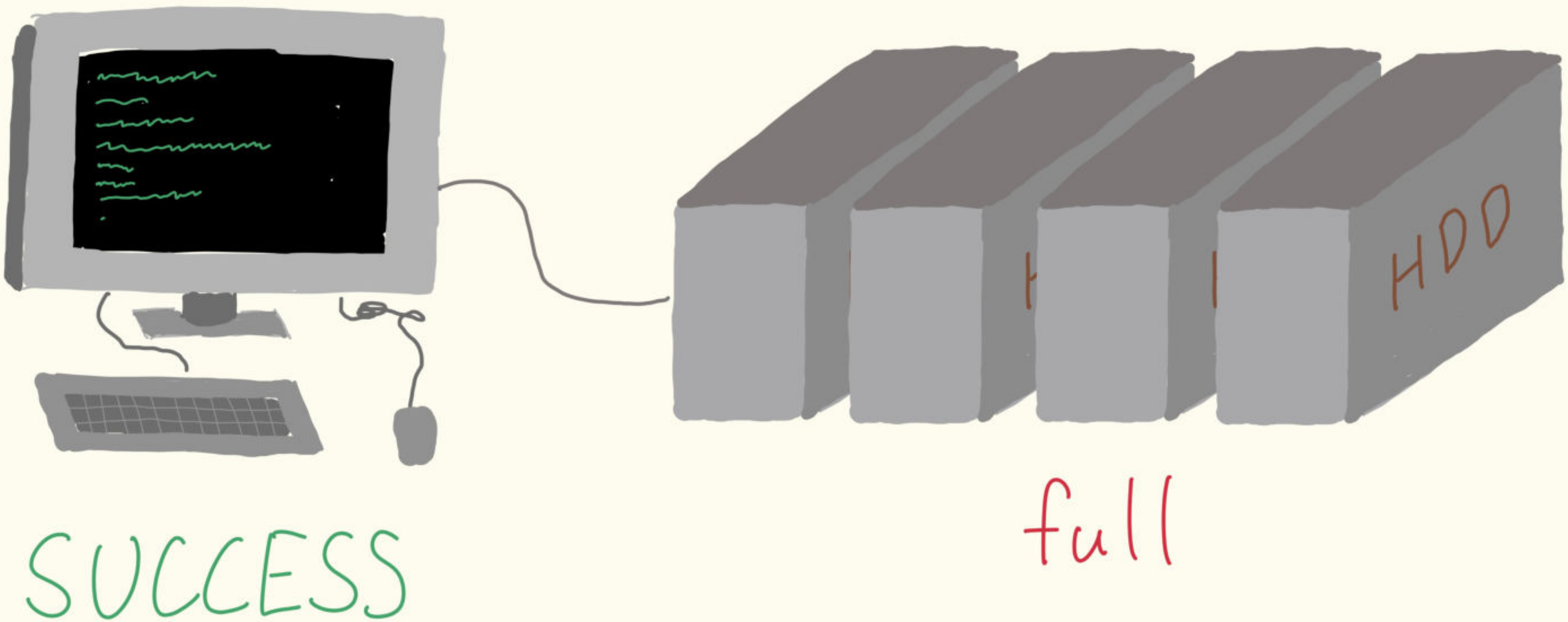


ERR : OUT_OF_MEM

THEORY OF COMPUTATION

Example : TIME vs SPACE

one "trivial" fact :

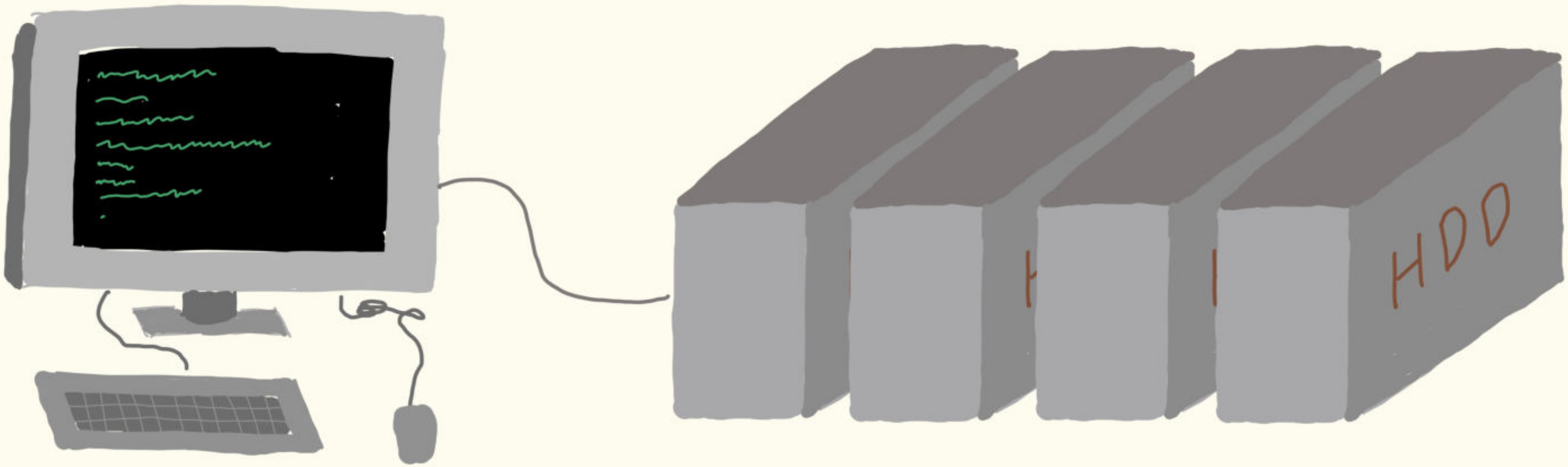


THEORY OF COMPUTATION

Example : TIME vs SPACE

one "trivial" fact :

NO WAY



SUCCESS

full

THEORY OF COMPUTATION

Example : TIME vs SPACE

one "~~trivial~~" fact :
false

[BCKLS'14] :

for many f
and every z

THEORY OF COMPUTATION

Example: TIME vs SPACE

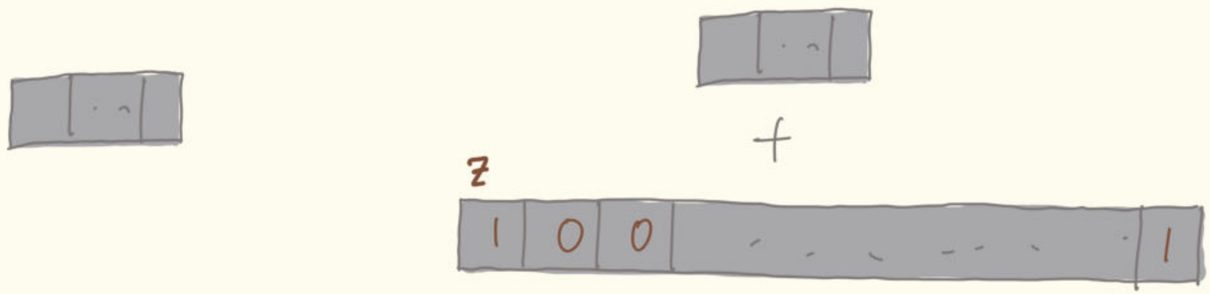
one "~~trivial~~" false fact:

$$\text{SPACE}(f, z) \ll \begin{array}{c}\text{SPACE}(f) \\ + \\ \text{SPACE}(z)\end{array}$$

[BCKLS'14]:

for many f and every z

$f \overset{?}{\in}$ but $f \in$



THEORY OF COMPUTATION

Example : TIME vs SPACE

one "~~trivial~~" fact :
false

[BCKLS'14]:

for many f
and every z

full memory provably adds a lot of power!

THEORY OF COMPUTATION

Example: TIME vs SPACE

Goal: TIME[2^t] requires SPACE[$\gg t$]

[CMWBS'12]: $\boxed{f^* \text{ in TIME}[2^t] \text{ but not SPACE}[<t^2]}$

Need to show:

"obvious" algorithm for f^* is space-optimal

[BCKLS'14]

"trivial" facts

Toy cases: ✓ ✓ ✓

Theory of Computation

Example: TIME vs SPACE

Goal: TIME[2^t] requires SPACE[$\gg t$]

~~[CMWBS'42]~~:

~~Need to show~~:

$\boxed{f^* \text{ in SPACE}[t \log t]}$

[CM'20,21,24]

[BCKLS'14]

Toy cases:

Theory of Computation

Example: TIME vs SPACE

Goal: TIME[2^t] requires SPACE[$\gg t$]

~~[CMWBS'12]~~: $\boxed{\text{SPACE}[t] \text{ requires } \text{TIME}[\gg t]}$ [W'25]

~~Need to show~~: $\boxed{f^* \text{ in SPACE}[t \log t]}$ [CM'20,21,24]

[BCKLS'14]

Toy cases:

[HPV'77]

THEORY OF COMPUTATION

Example : TIME vs SPACE

$$\mathrm{TIME}[t] \subseteq \mathrm{SPACE}[t] \subseteq \mathrm{TIME}[2^t]$$

Twist

f in SPACE[t]
but not TIME[t]

Goal

f in TIME[2^t]
but not SPACE[t]

led us to ideas for

Result

key piece needed for

f^* in SPACE[$t \log t$]

THEORY OF COMPUTATION

Example : TIME vs SPACE

$$\mathrm{TIME}[t] \subseteq \mathrm{SPACE}[t] \subseteq \mathrm{TIME}[2^t]$$

Twist

f in $\mathrm{SPACE}[t]$
but not $\mathrm{TIME}[t]$

Goal

f in $\mathrm{TIME}[2^t]$
but not $\mathrm{SPACE}[t]$

led us to ideas for

Result

f^* in $\mathrm{SPACE}[t \log t]$

key piece needed for

Mega-twist

every f in $\mathrm{TIME}[t]$
is in $\mathrm{SPACE}[\sqrt{t \log t}]$

implies

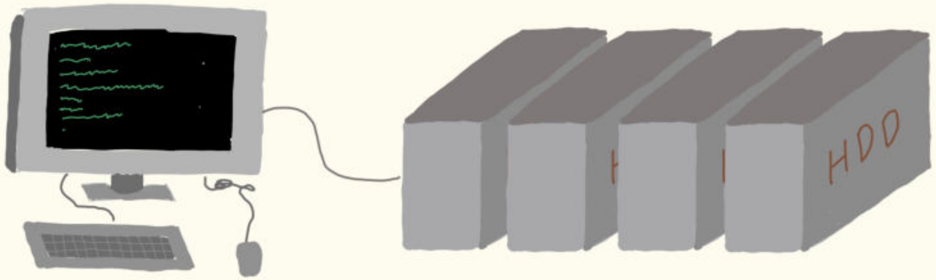
[W'25]

Theory of Computation

Example: TIME vs SPACE

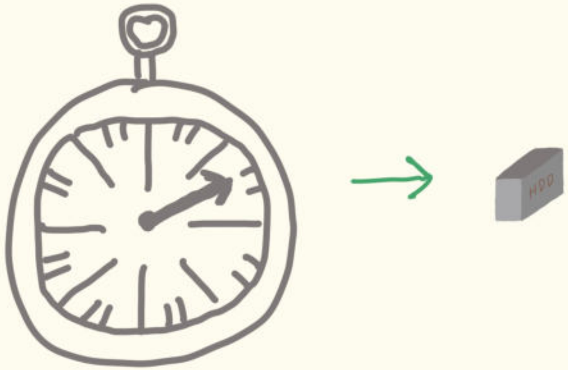
$$\text{SPACE}(f, z) \ll \text{SPACE}(f) + \text{SPACE}(z)$$

every f in $\text{TIME}[t]$ is in $\text{SPACE}[\sqrt{t \log t}]$



SUCCESS

full



Theory of Computation

Algorithms

"upper bounds"

- study of what is possible
- chiefly limited by human ingenuity (or lack thereof)
- can translate to practical implementations

Complexity

"lower bounds"

- study of what is impossible
- need to disprove <u>every</u> algorithm (discovered or not)
- tells us when to look for a different approach

Theory of Computation

Algorithms

"upper bounds"

motivates

creates

Complexity

"lower bounds"

- study of what is possible
- chiefly limited by human ingenuity (or lack thereof)
- can translate to practical implementations

- study of what is impossible
- need to disprove every algorithm (discovered or not)
- tells us when to look for a different approach

THEORY OF COMPUTATION

TIME

- speed of computation

ONCE IT'S USED, GONE FOREVER

power of randomness, etc.

SPACE

- computing under memory

CAN BE ERASED AND REUSED

ted

ting

- sc

but very different answers!

THEORY OF COMPUTATION

TIME

- speed of computation

ONCE IT'S USED, GONE FOREVER

power of randomness, etc.

SPACE

- computing under memory

ted

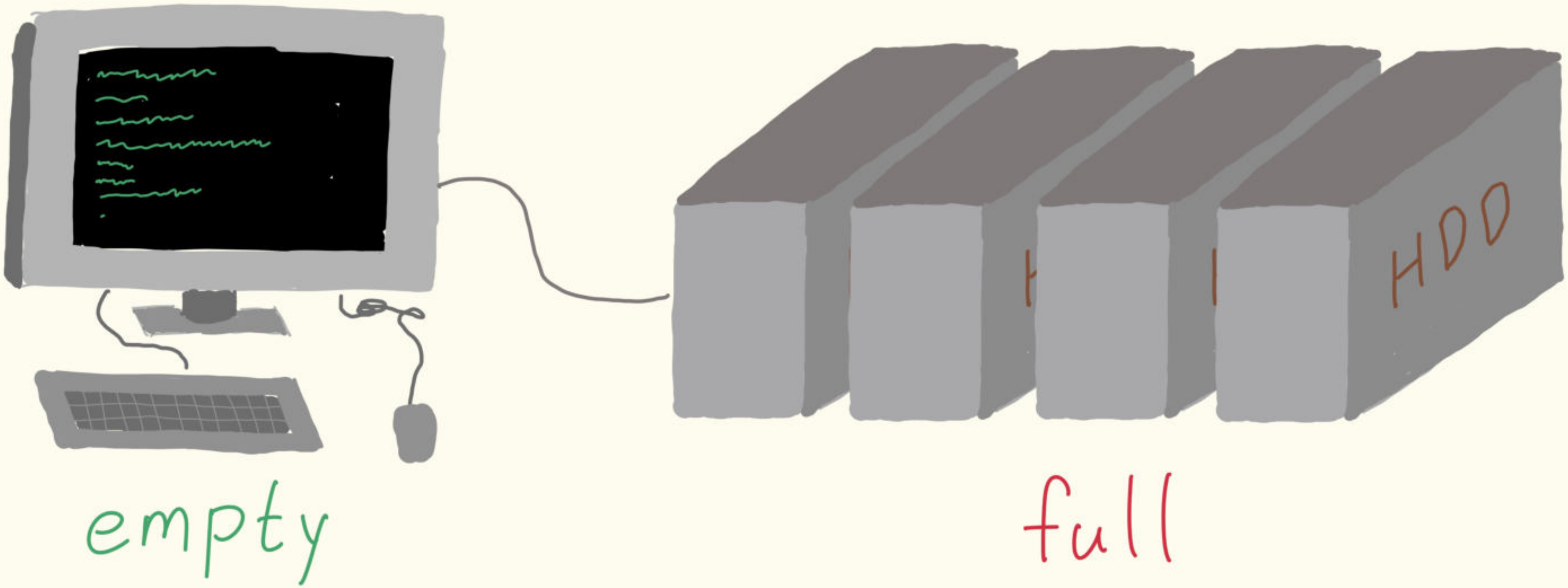
ting

CAN BE ~~ERASED~~ ~~AND~~ REUSED <u>MANY</u> WAYS!

s!

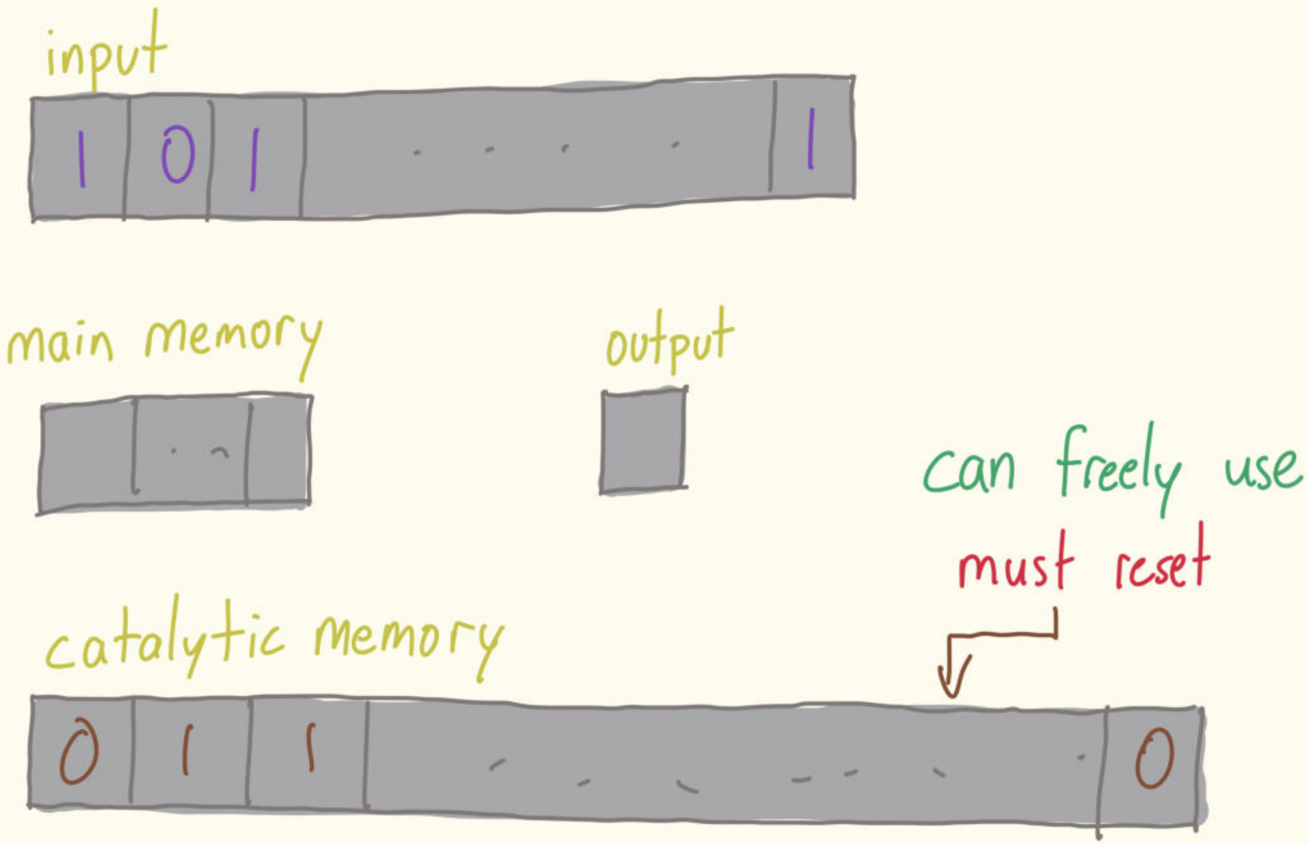
But What Can You <u>Actually</u> Do With A Full Hard Drive?

THE STUDY OF REUSE



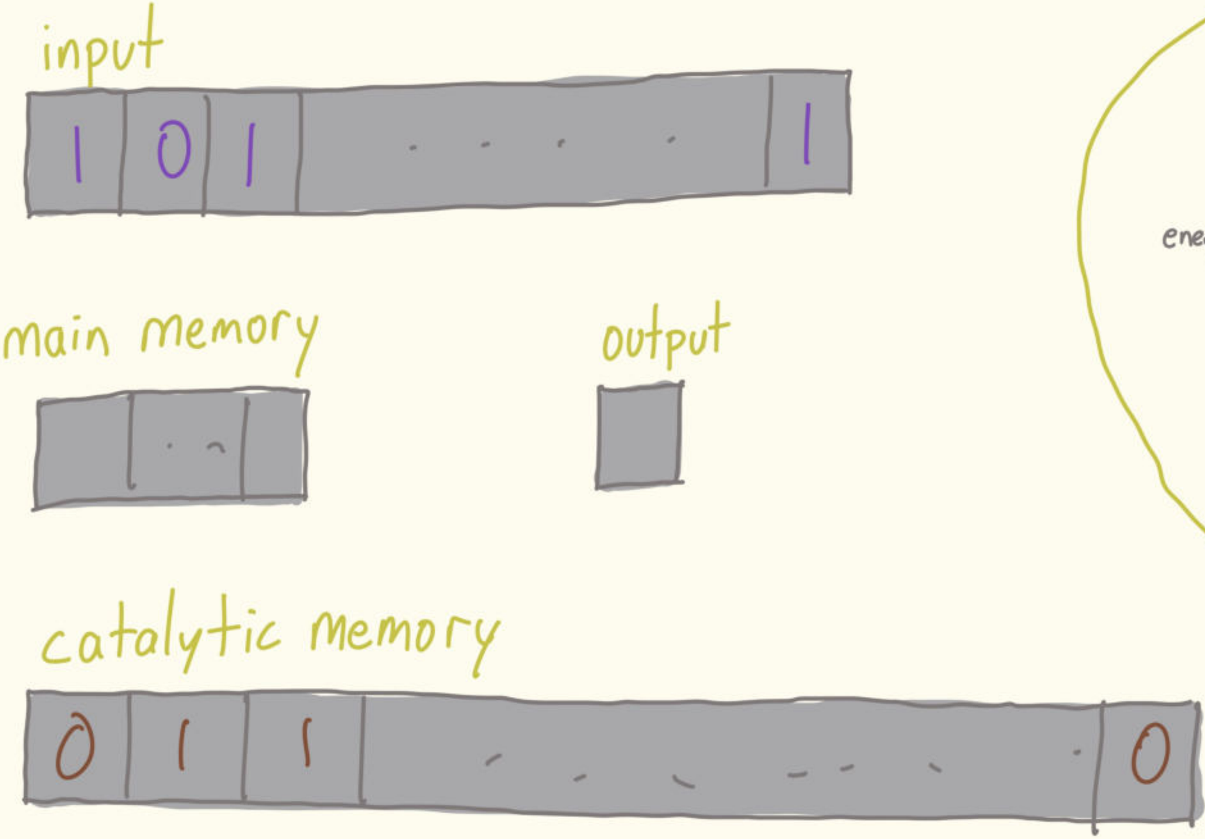
The Study of Reuse

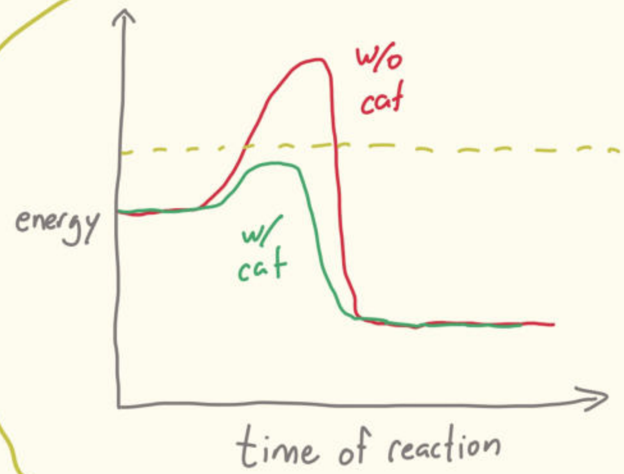
catalytic computing [BCKLS'14]



The Study of Reuse

catalytic computing [BCKLS'14]





$2NO + O_2 \rightarrow 2NO_2$

$NO_2 + SO_2 \rightarrow NO + SO_3$

Net: $\underset{+2NO}{2SO_2} + O_2 \rightarrow \underset{+2NO}{2SO_3}$

Two Key Ideas

1. Compression

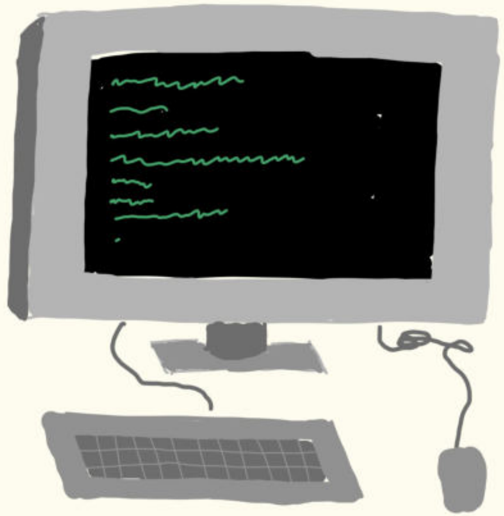
2. Reversibility

Two Key Ideas

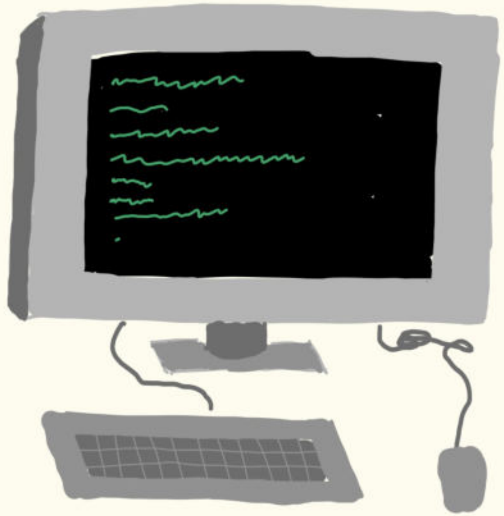
1. Compression

2. Reversibility

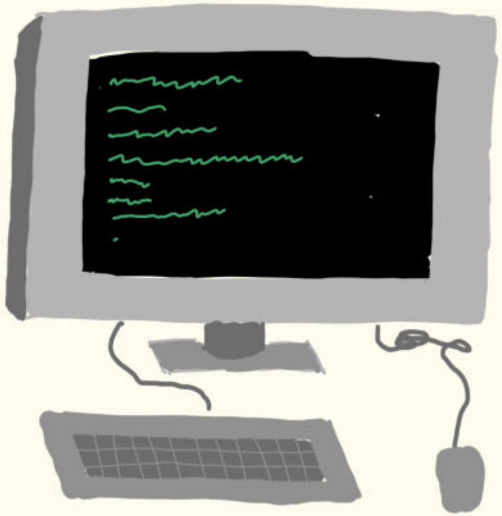
Compress



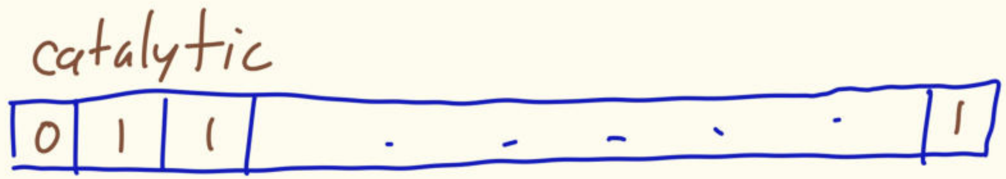
~~Compress~~



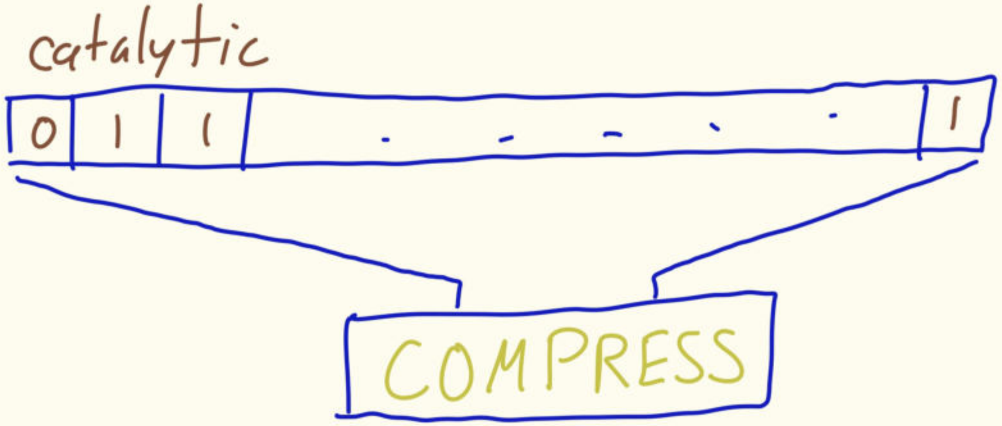
~~Compress~~ Random



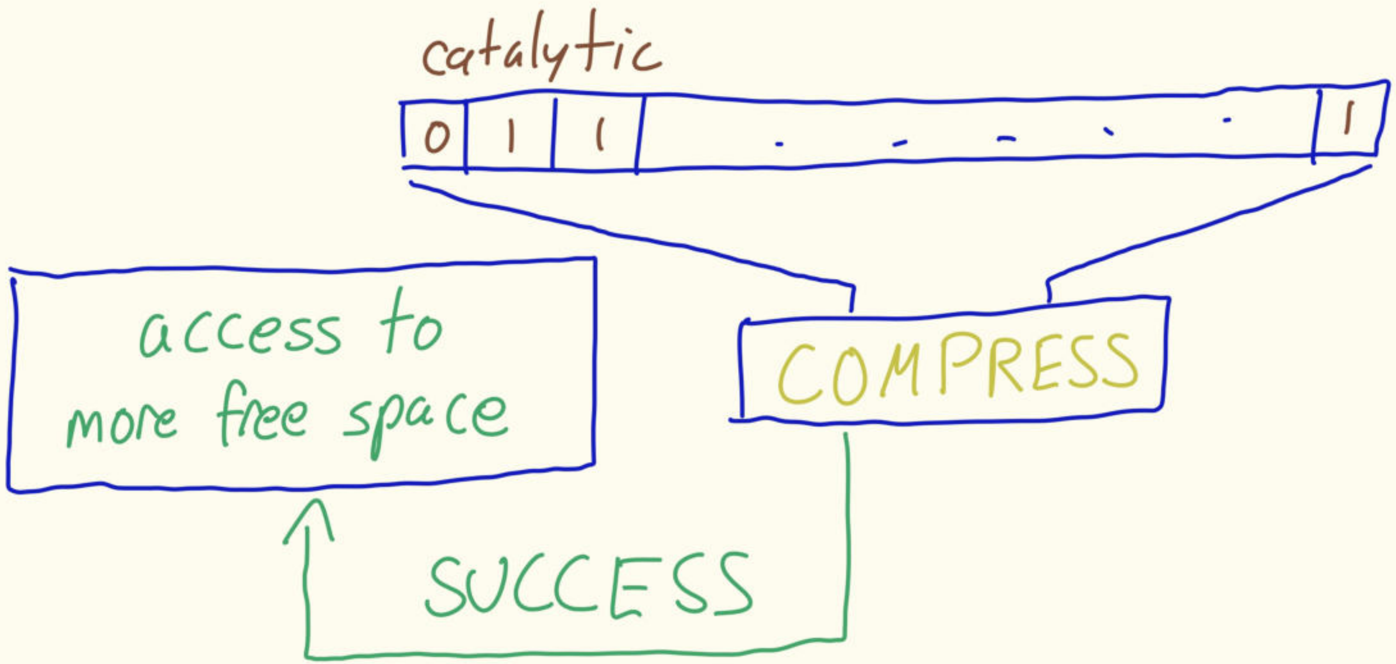
Compress-or-Random



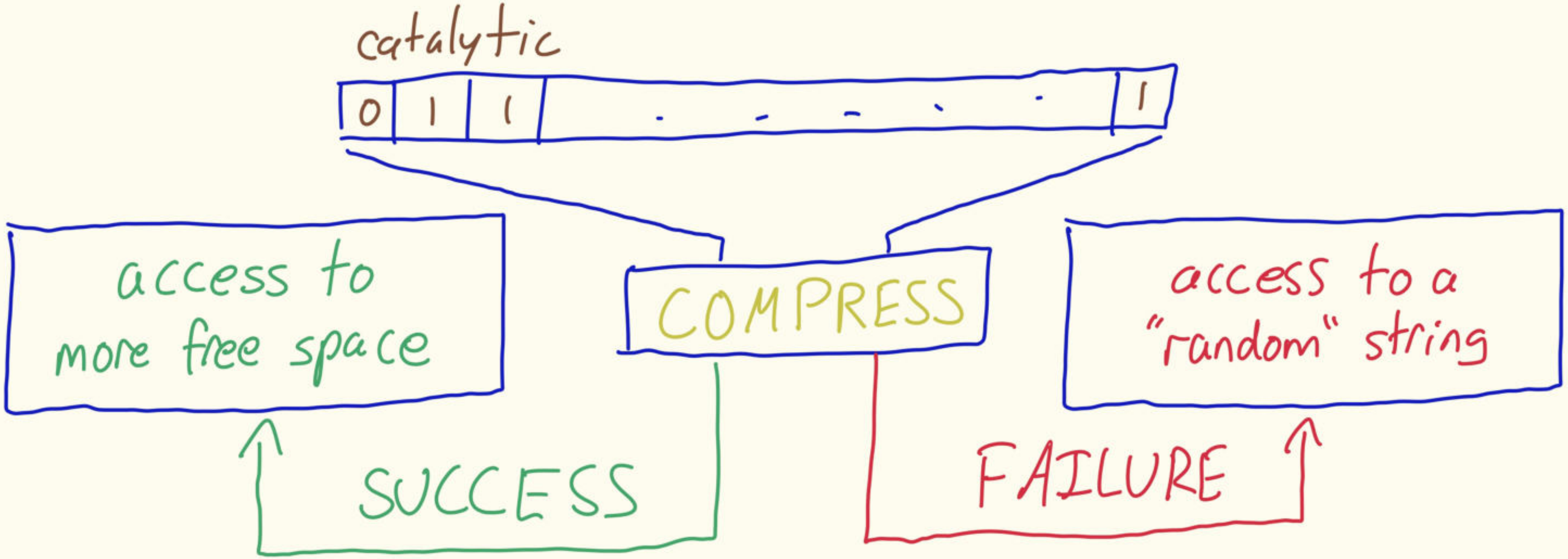
Compress-or-Random



Compress-or-Random

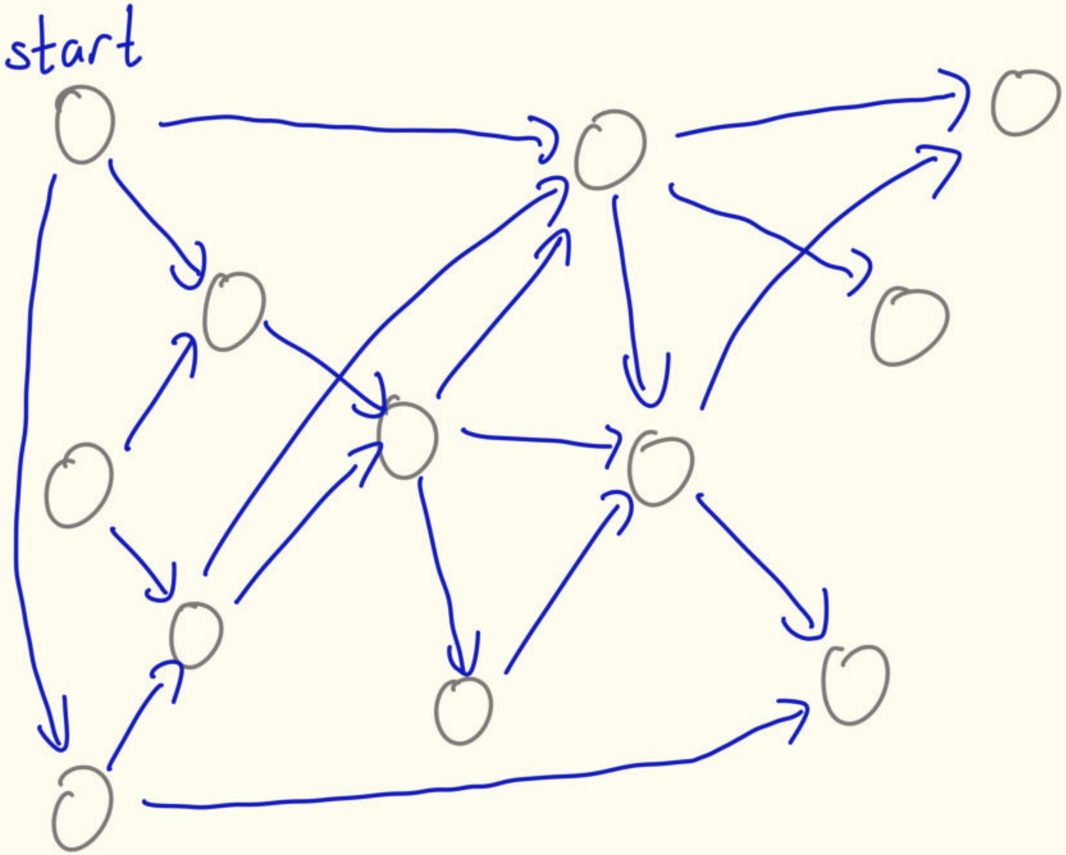


Compress-or-Random



Compress-or-Random

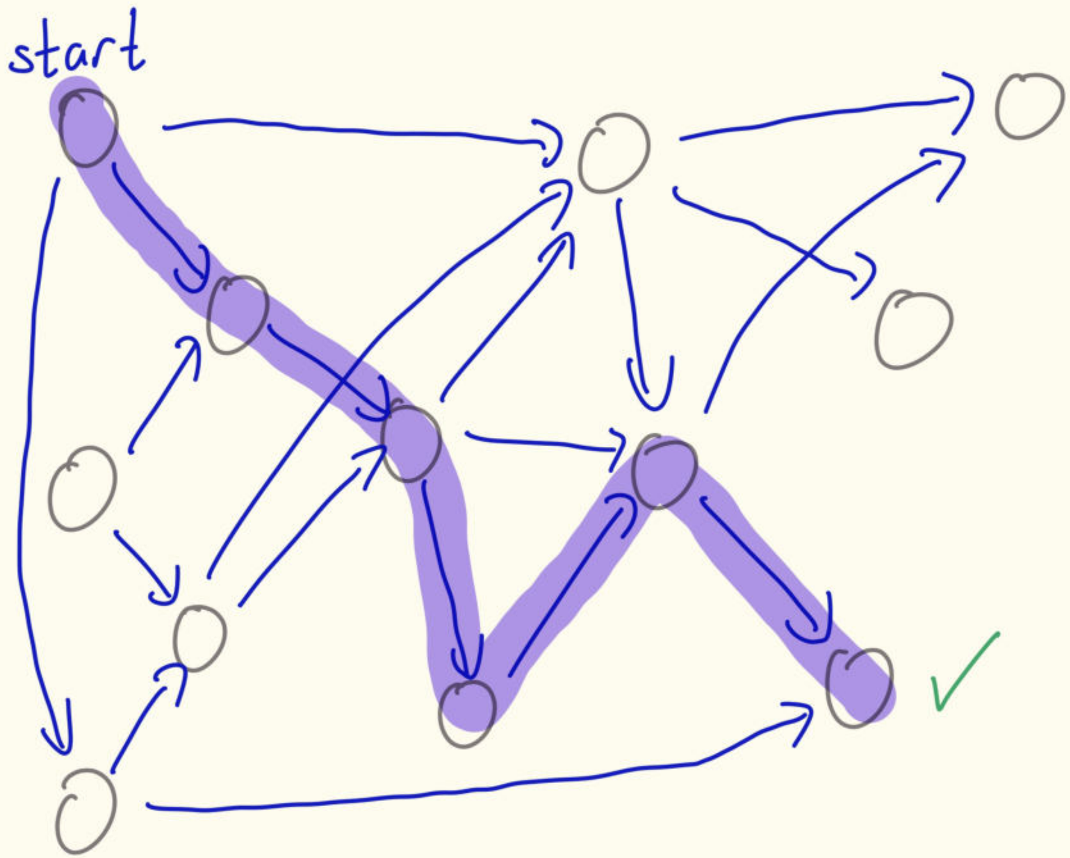
Example: random walks



assumption:
$\geq 2/3$ of all walks from start go to the same final node

Compress-or-Random

Example: random walks

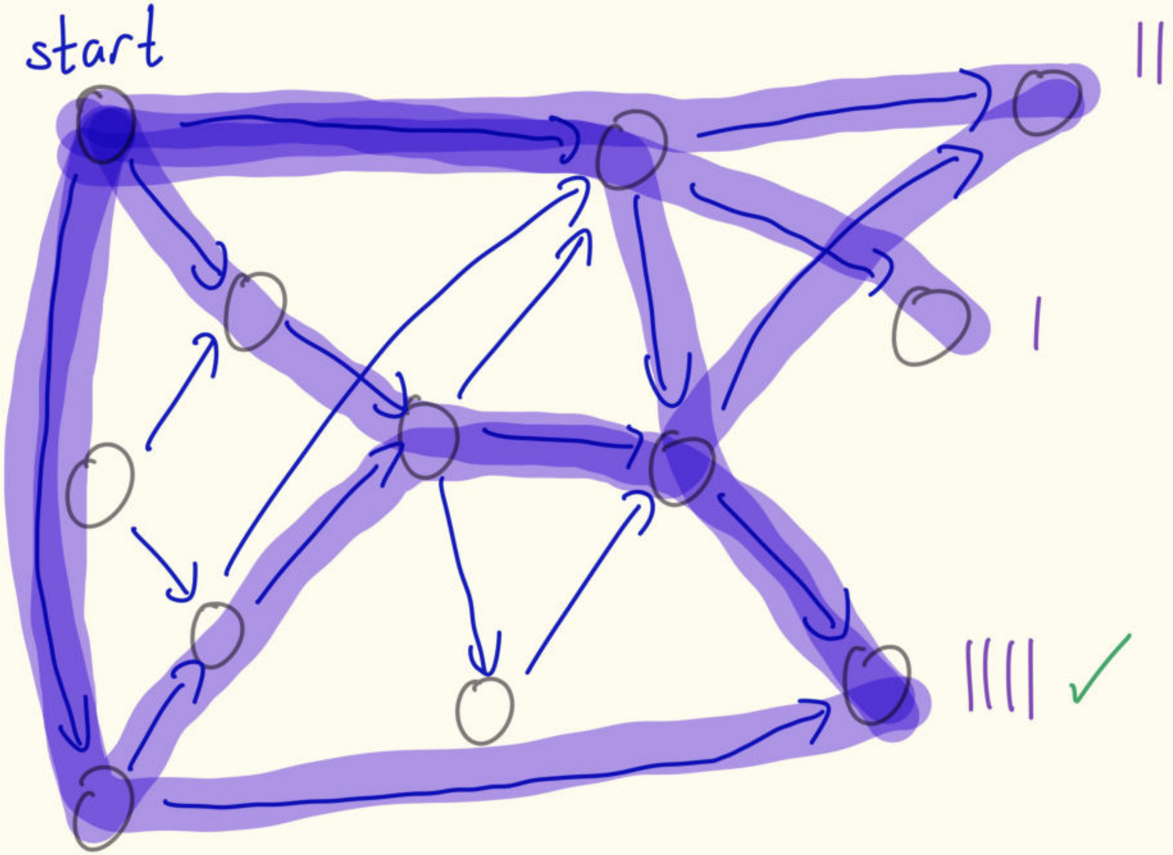


assumption:
$\geq 2/3$ of all walks from start go to the same final node

$\longrightarrow$ a random walk goes there with $\geq 2/3$ probability

Compress-or-Random

Example: random walks



many walks →
odds of right answer
boosted exponentially

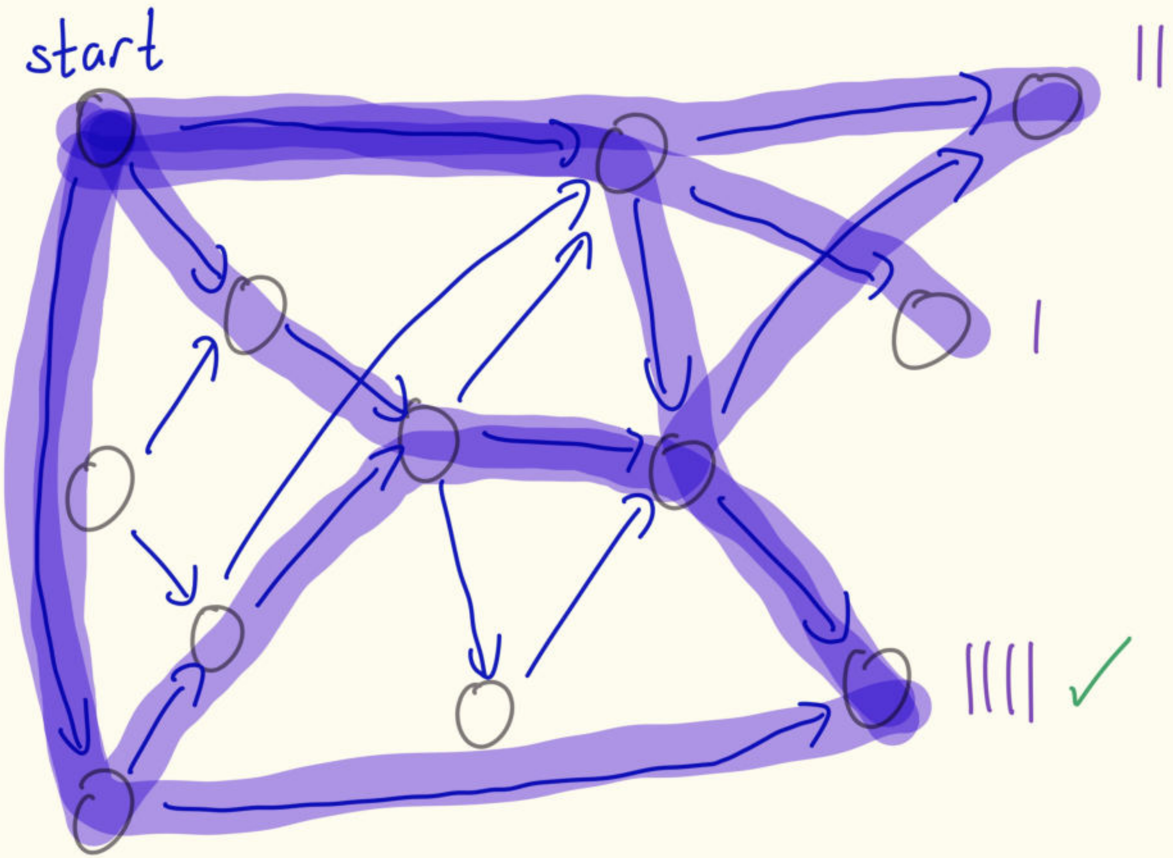
random

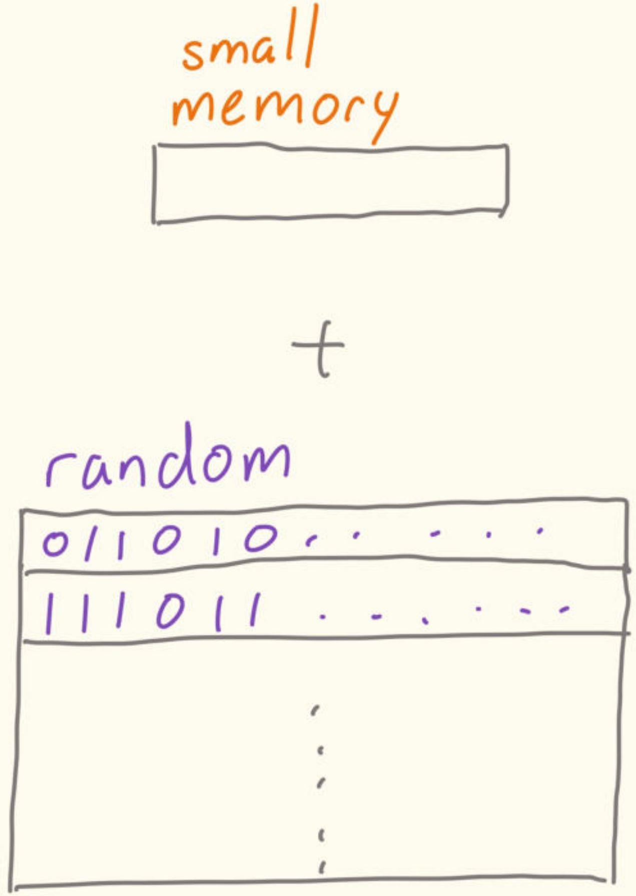
011010 . . . . . .
111011 . . . . . .
⋮

COMPRESS-OR-RANDOM

only need to remember current node in walk (plus tally of answers)

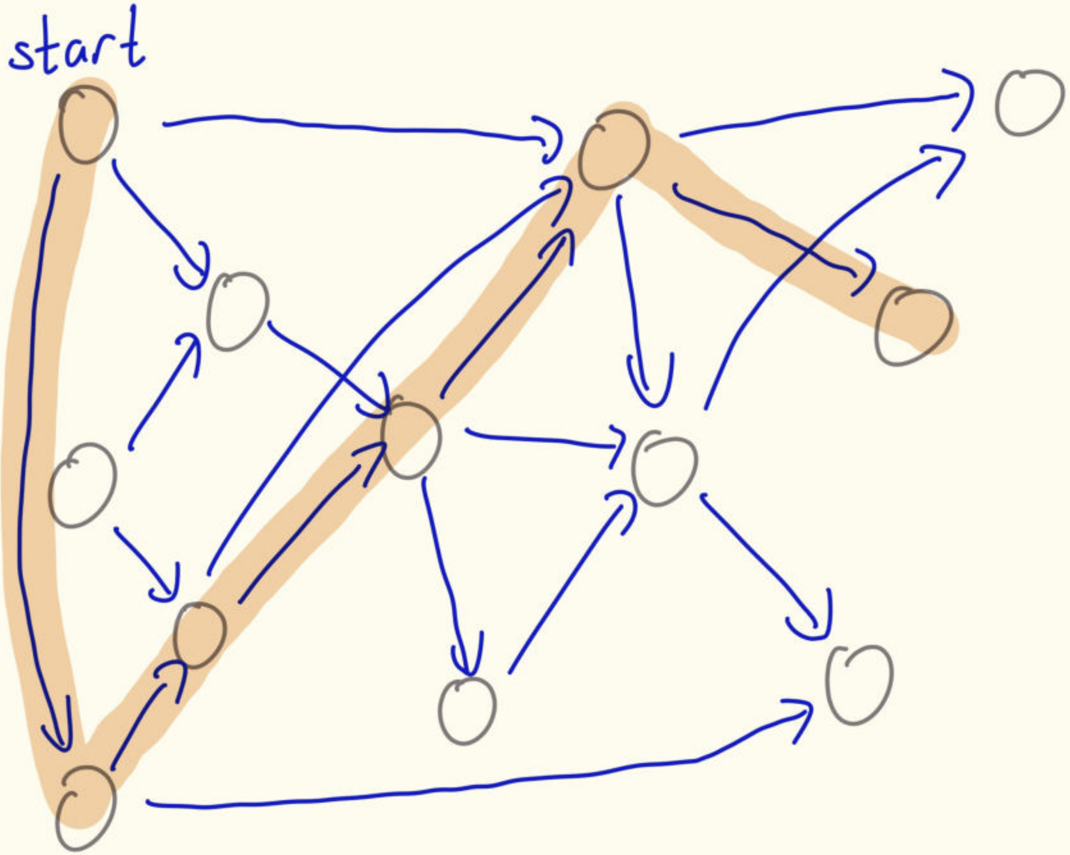
Example: random walks





Compress-or-Random

Example: random walks



small memory

+

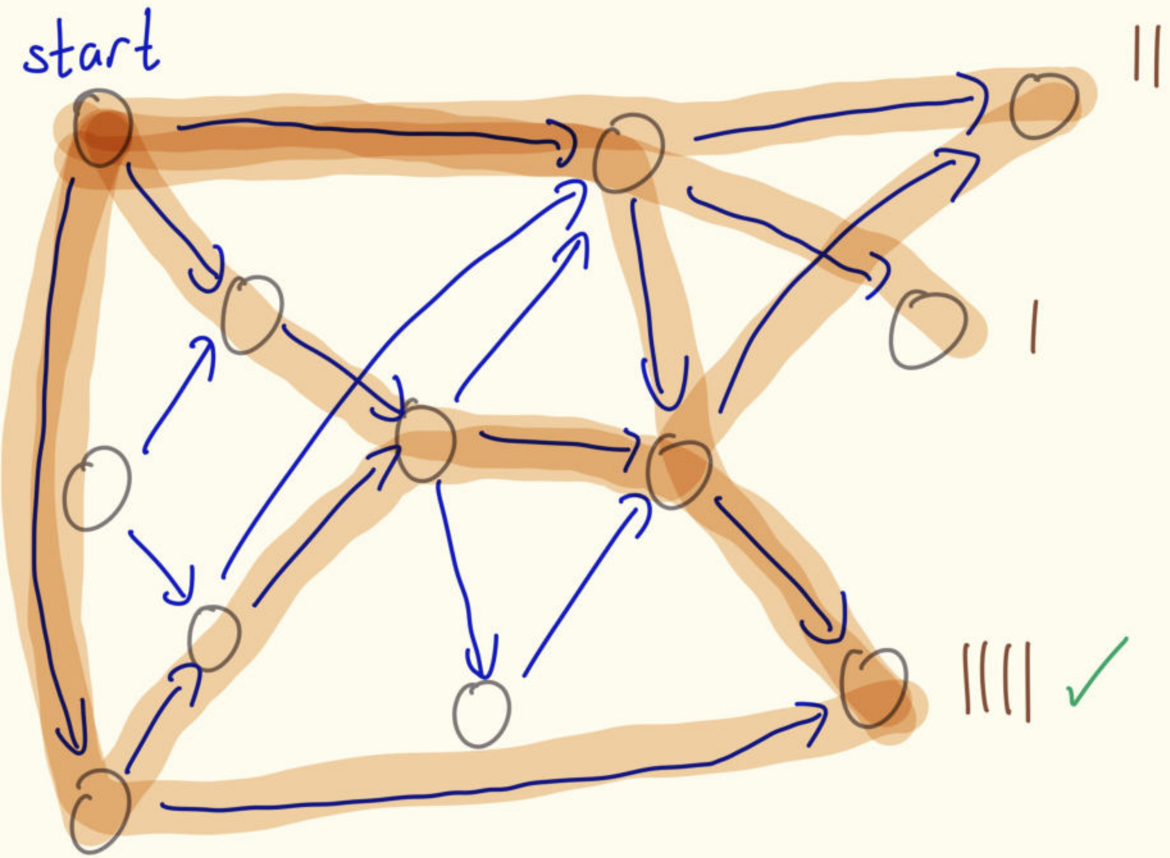
catalytic

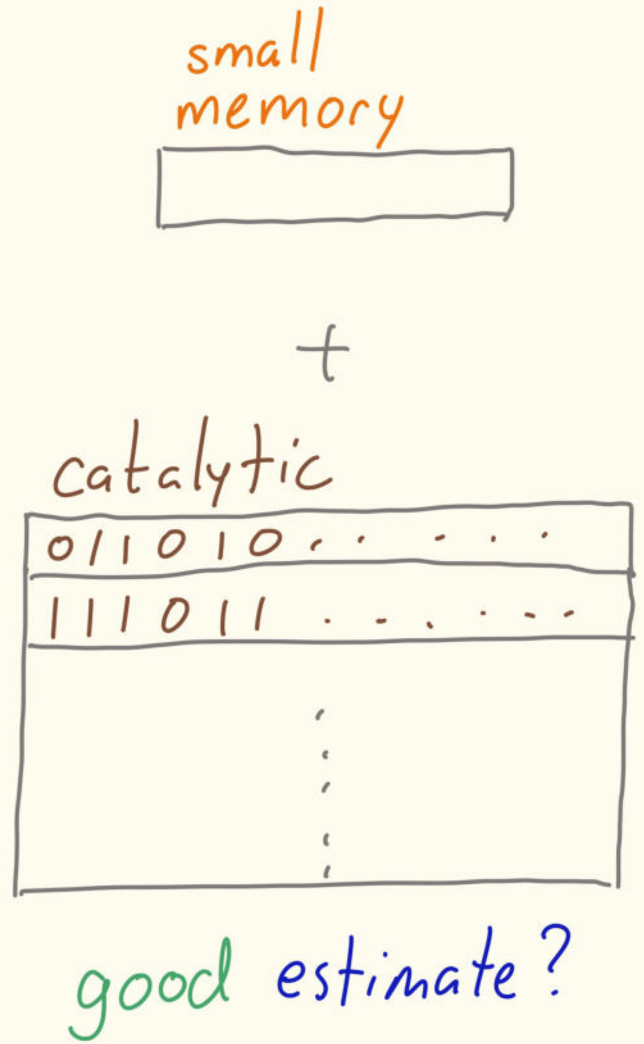
0 1 1 0 1 0 . . . . . .

good walk ?

Compress-or-Random

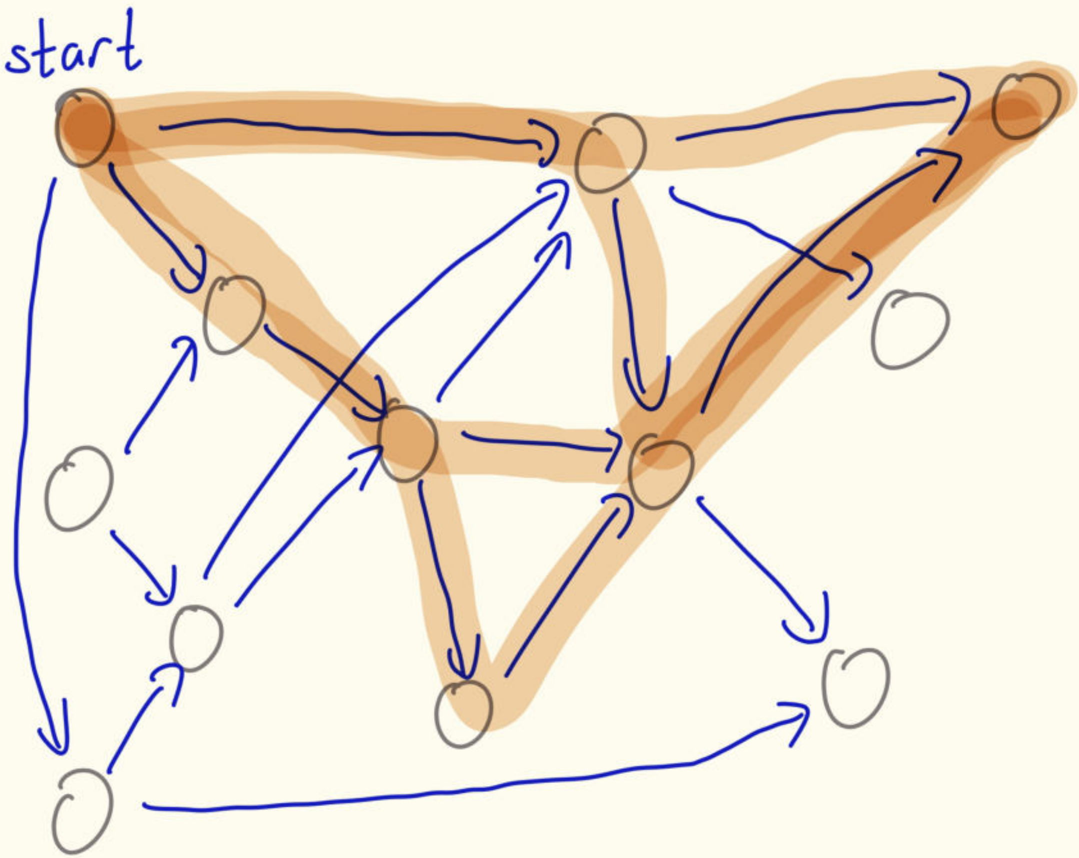
Example: random walks

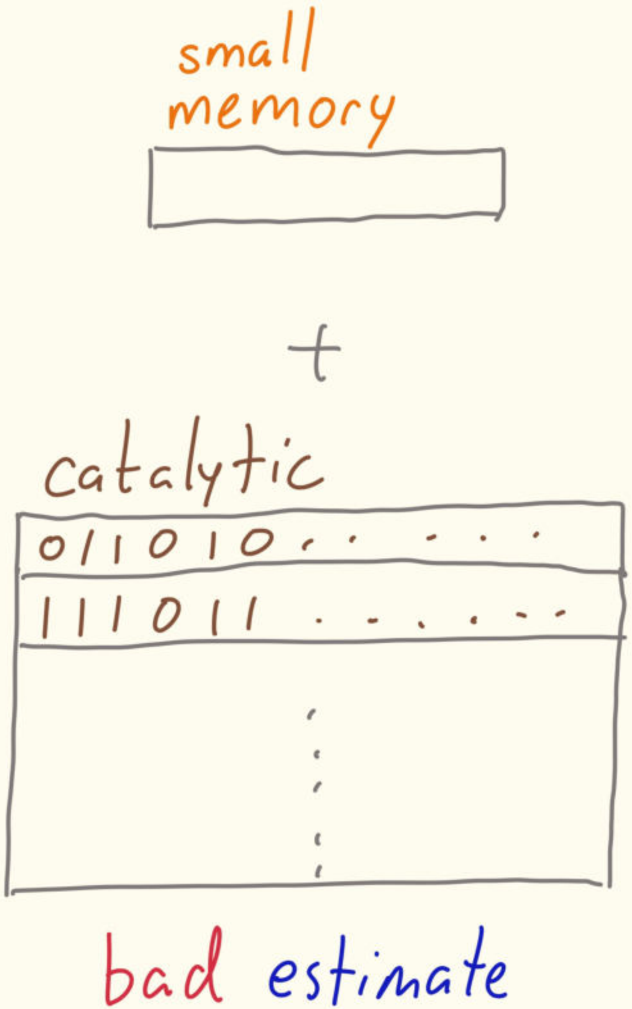




Compress-or-Random

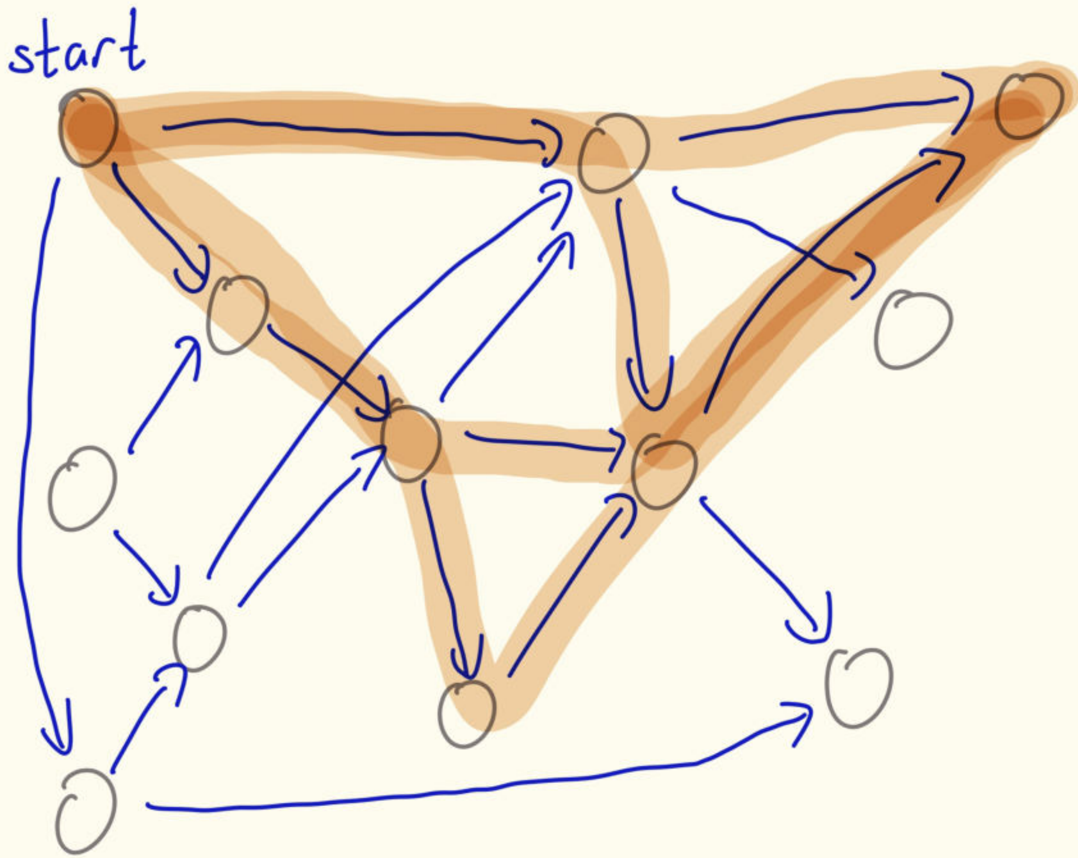
Example: random walks

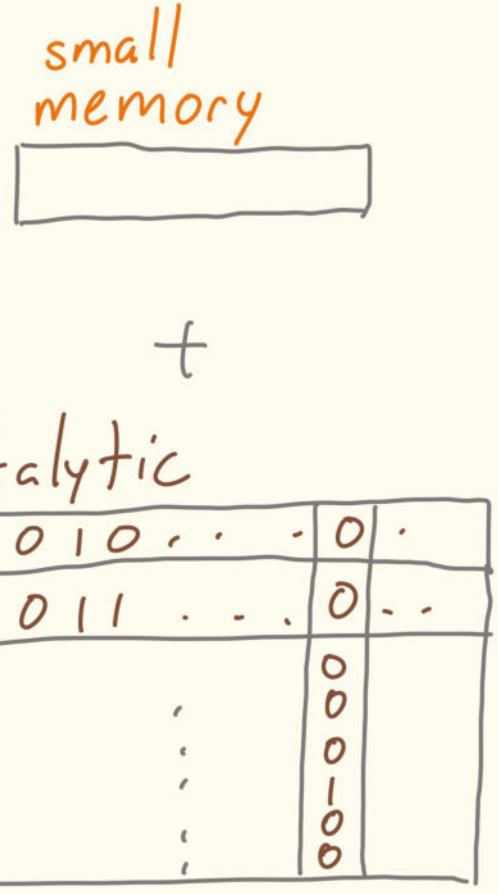




Compress-or-Random

Example: random walks



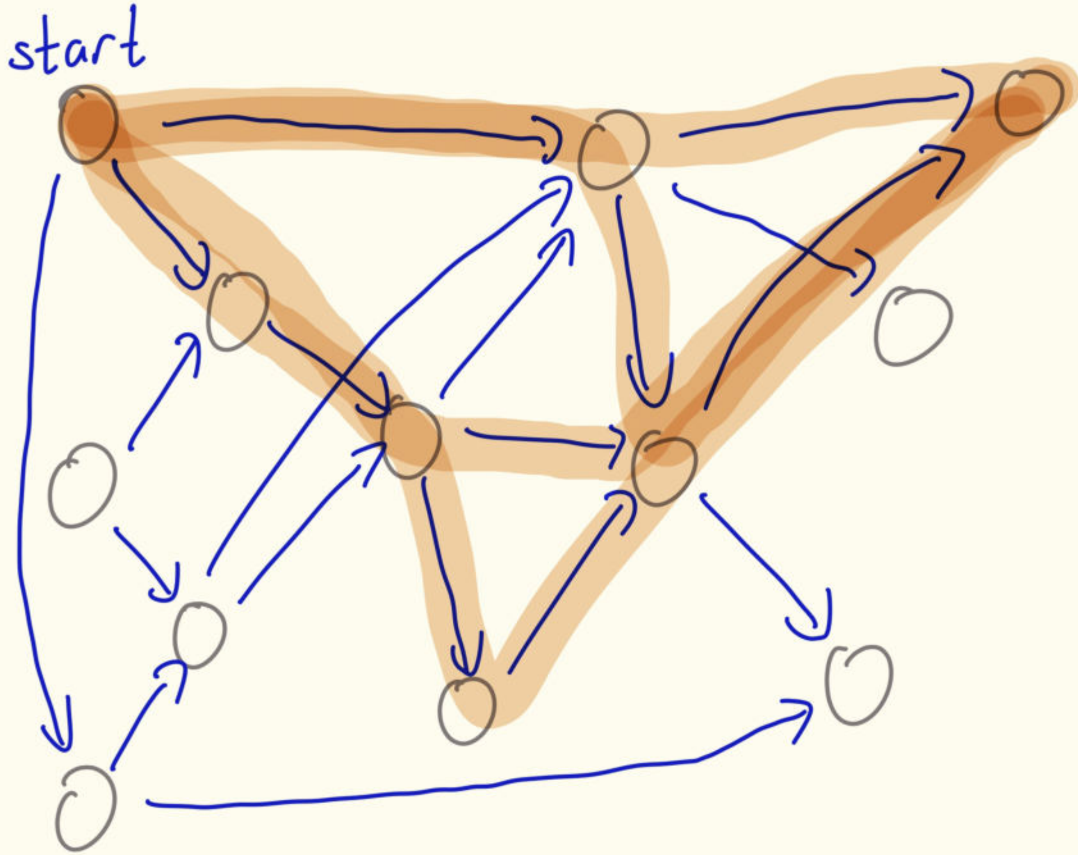


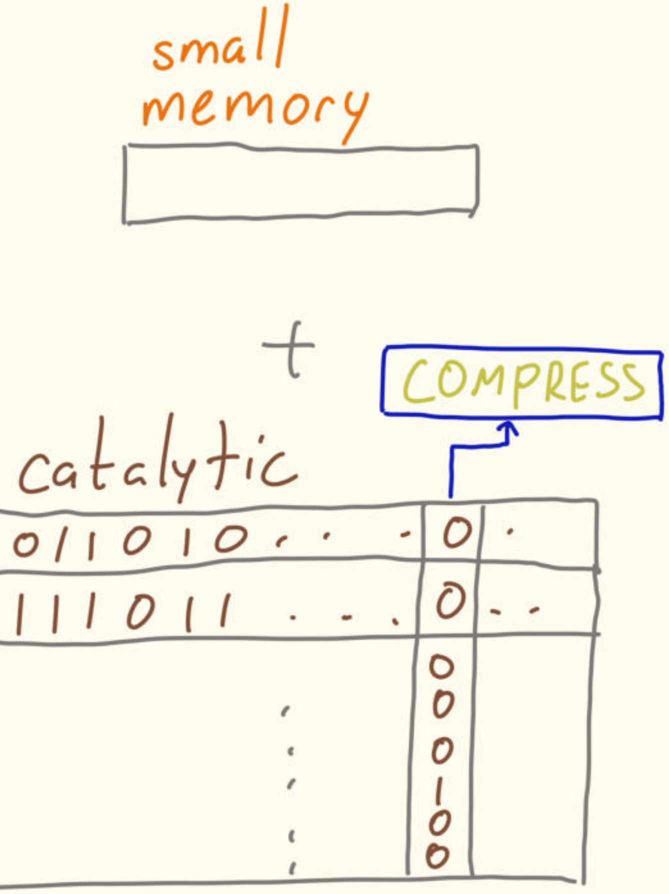
[N'92]: bad estimate →

massively biased subsection

Compress-or-Random

Example: random walks





[N'92]: bad estimate → massively biased subsection

Two Key Ideas

1. Compress-or-Random ✓

2. Reversibility

Two Key Ideas

1. Compress-or-Random ✓

2. Reversibility

Arithmetic Magic

Arithmetic Magic

Example: swapping memory

Arithmetic Magic

Example: swapping memory

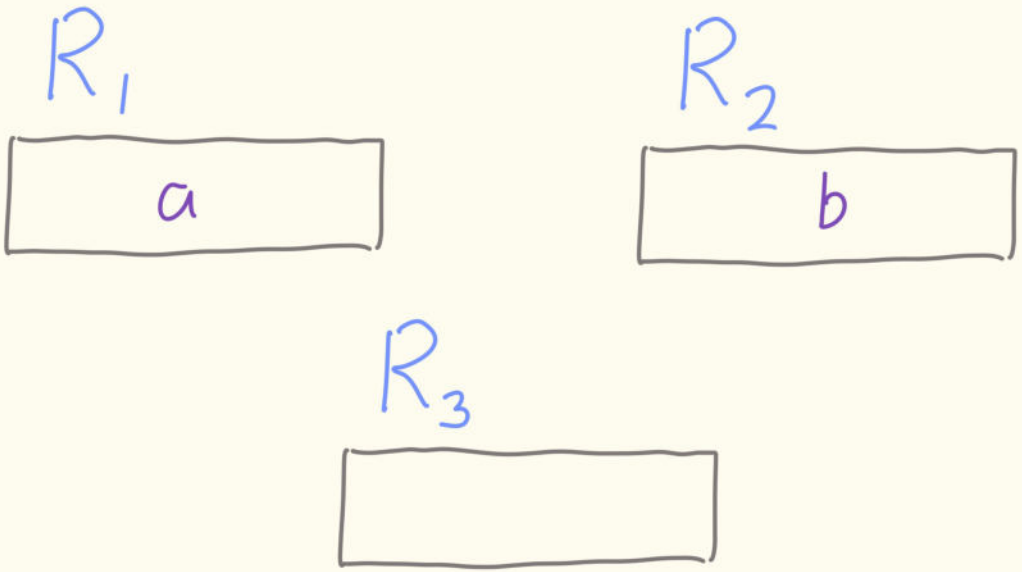
$$\text{Given: } \quad \overset{R_1}{\boxed{\quad a \quad}} \qquad \overset{R_2}{\boxed{\quad b \quad}}$$

$$\downarrow$$

$$\text{Want: } \quad \overset{R_1}{\boxed{\quad b \quad}} \qquad \overset{R_2}{\boxed{\quad a \quad}}$$

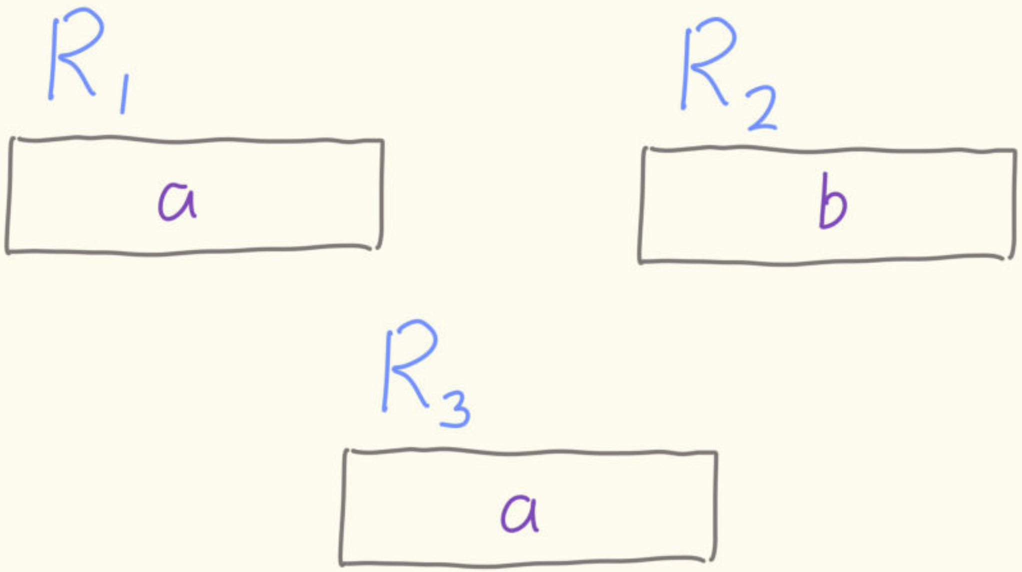
ARITHMETIC MAGIC

Example: swapping memory



Arithmetic Magic

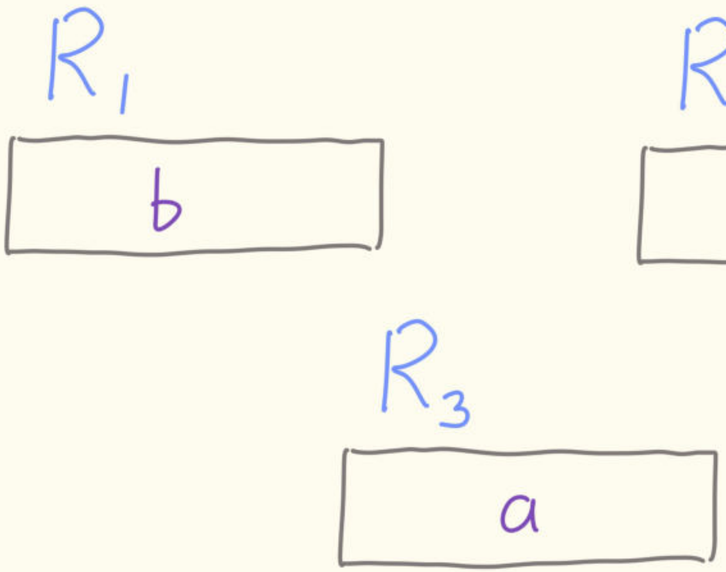
Example: swapping memory



1. $R_3 = R_1$

ARITHMETIC MAGIC

Example: swapping memory

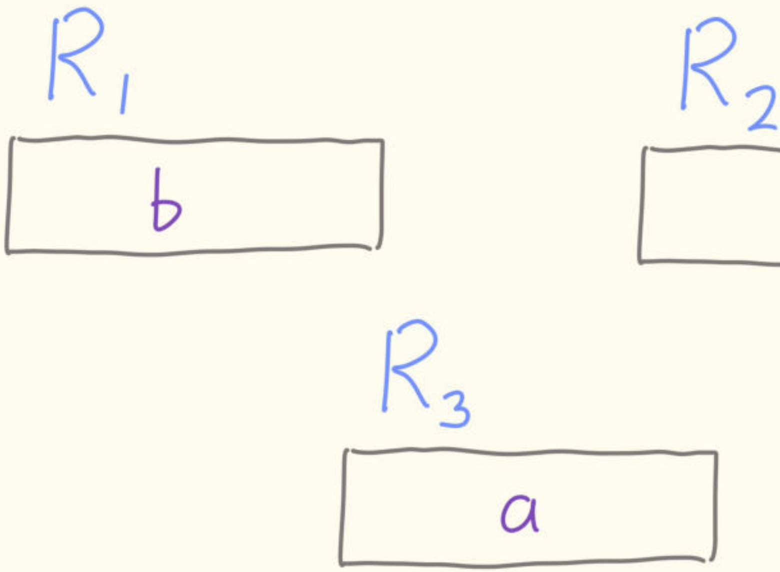


1. $R_3 = R_1$

2. $R_1 = R_2$

ARITHMETIC MAGIC

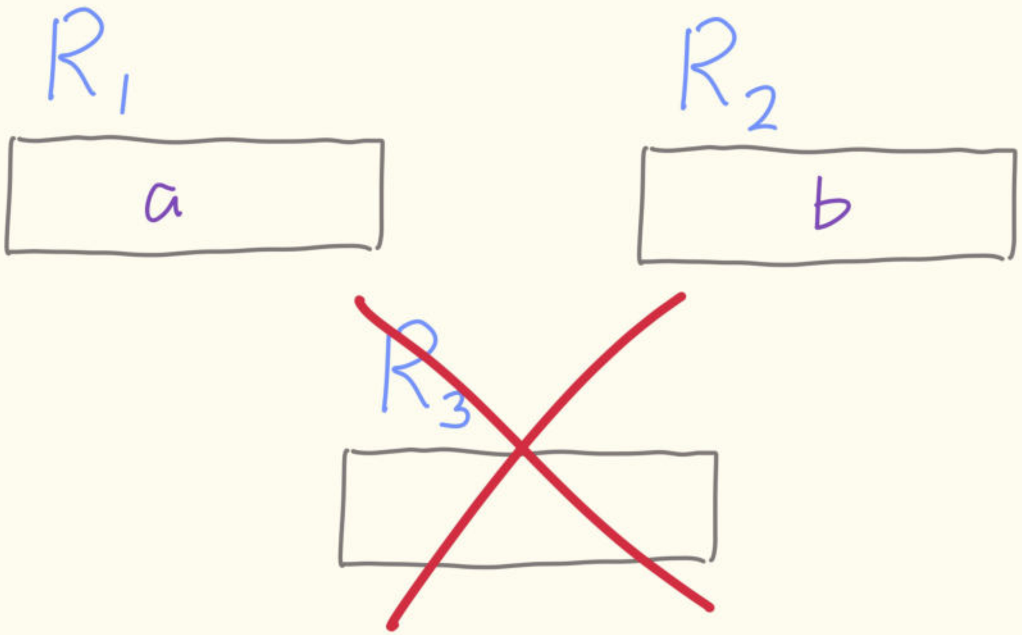
Example: swapping memory



1. $R_3 = R_1$
2. $R_1 = R_2$
3. $R_2 = R_3$

ARITHMETIC MAGIC

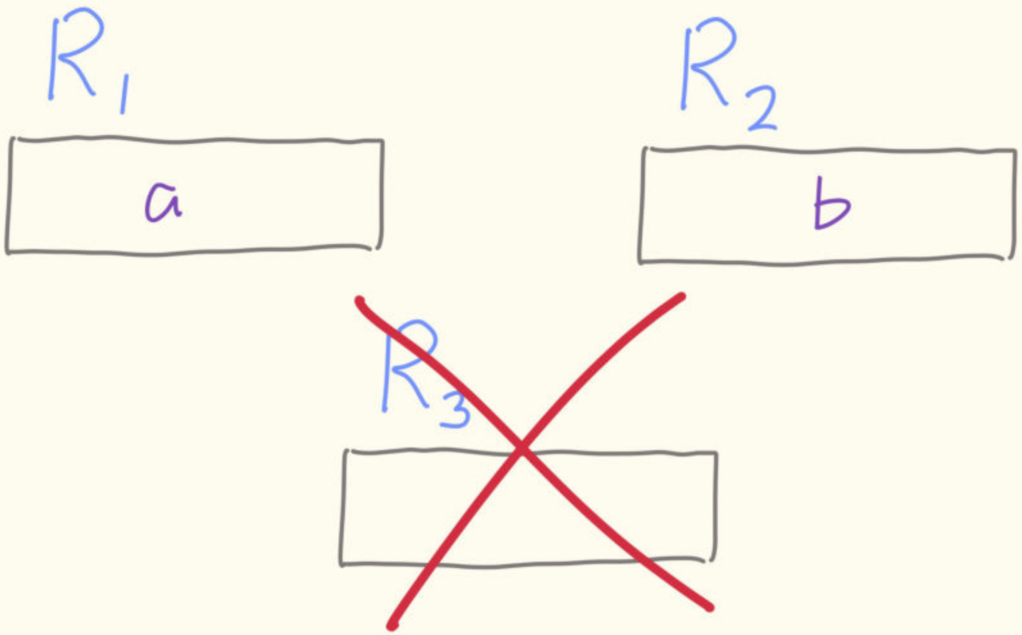
Example: swapping memory



Arithmetic Magic

for simplicity:
a, b numbers

Example: swapping memory



ARITHMETIC MAGIC

for simplicity:
a, b numbers

Example: swapping memory

1. $R_1 = R_1 - R_2$

ARITHMETIC MAGIC

for simplicity:
a, b numbers

Example: swapping memory

1. $R_1 = R_1 - R_2$

ARITHMETIC MAGIC

for simplicity:
a, b numbers

Example: swapping memory

1. $R_1 = R_1 - R_2$
2. $R_2 = R_1 + R_2$

ARITHMETIC MAGIC

for simplicity:
a, b numbers

Example: swapping memory

R_1: $\boxed{a-b}$

R_2: $\boxed{(a-b)+b}$ $=a$

1. $R_1 = R_1 - R_2$
2. $R_2 = R_1 + R_2$

ARITHMETIC MAGIC

for simplicity:
a, b numbers

Example: swapping memory

1. $R_1 = R_1 - R_2$
2. $R_2 = R_1 + R_2$
3. $R_1 = R_2 - R_1$

ARITHMETIC MAGIC

for simplicity: a, b numbers

Example: swapping memory

R_1: $\boxed{a-(a-b)}$ $= b$

R_2: $\boxed{a}$

1. $R_1 = R_1 - R_2$
2. $R_2 = R_1 + R_2$
3. $R_1 = R_2 - R_1$

ARITHMETIC MAGIC

for simplicity:
a, b numbers

Example: swapping memory

1. $R_1 = R_1 - R_2$
2. $R_2 = R_1 + R_2$
3. $R_1 = R_2 - R_1$

ARITHMETIC MAGIC

Don't store, recompute!

ARITHMETIC MAGIC

Don't store, recompute!

$\boxed{\quad \tau \quad}$, X

Arithmetic Magic

Don't store, recompute!

$$\boxed{\tau + x}$$

ARITHMETIC MAGIC

Don't store, recompute!

$$\boxed{\tau + x - x}$$

→ RESET

(x computed twice)

ARITHMETIC MAGIC

Don't store, recompute!

$\boxed{\quad\tau\quad}\ ,\ x\ ,\ y$

Goal: $xy \rightarrow$

[BC'92]

ARITHMETIC MAGIC

Don't store, recompute!

$\boxed{\tau + x}$

Arithmetic Magic

Don't store, recompute!

$$\boxed{\tau + x} \times y \xrightarrow{+} \boxed{\tau y + xy}$$

Arithmetic Magic

Don't store, recompute!

$$\boxed{\tau + x - x}$$

→ RESET

(x computed twice)

$$\boxed{\tau y + x y}$$

ARITHMETIC MAGIC

Don't store, recompute!

$$\boxed{\tau} \times y \qquad -$$

$$\boxed{\tau y + xy - \tau y}$$

→ RESET

(x computed twice)

Arithmetic Magic

Don't store, recompute!

$\boxed{\tau}$

x computed twice

$\boxed{xy}$

$\longrightarrow$ can use x without storing it directly

Two Key Ideas

1. Compress-or-Random ✓

2. Arithmetic Reversibility ✓

But What Can You Actually Do With A Full Hard Drive?

The History of Reuse





catalytic is born!
... but what else happened?

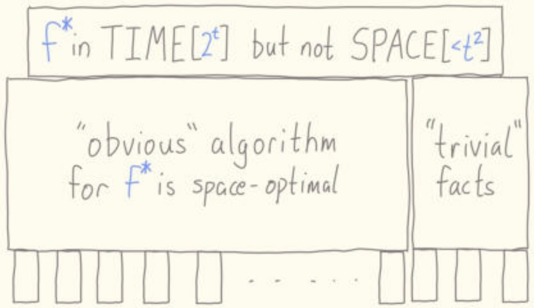
THE HISTORY OF REUSE

foundational results in space
contain seeds of catalytic ideas

The History of Reuse





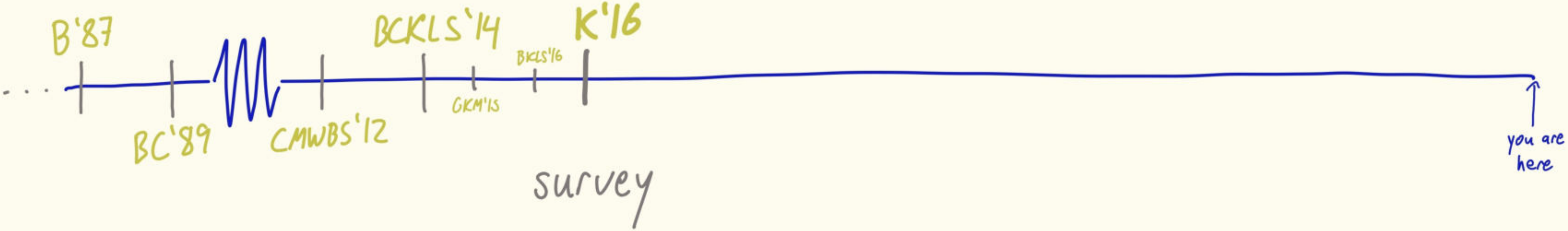


inspiration for catalytic

The History of Reuse

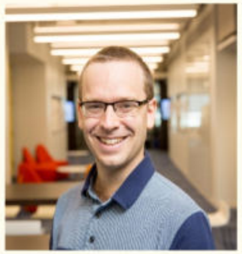
early theory
of catalytic computing

THE HISTORY OF REUSE



The History of Reuse







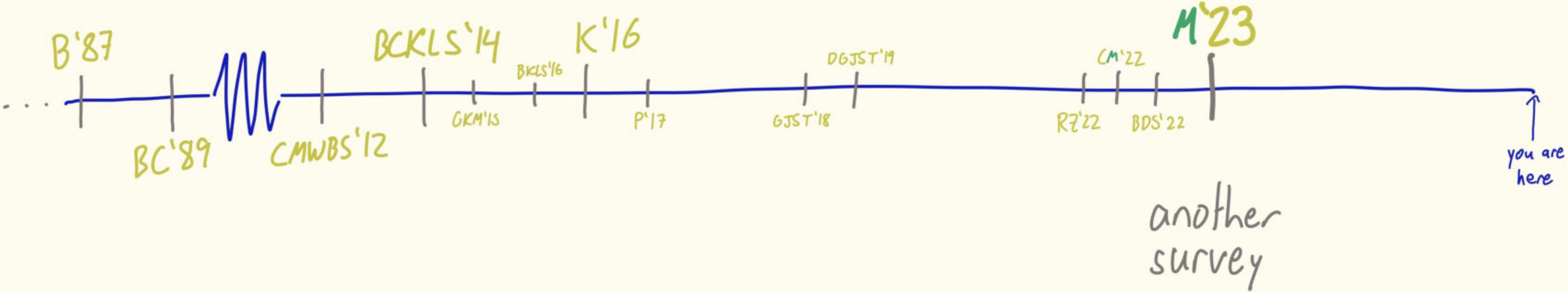




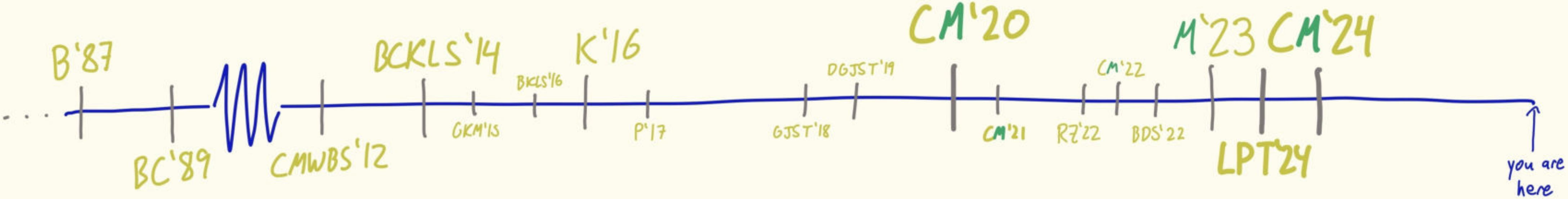


fleshing out theory,
defining more models,
many basic results

The History of Reuse

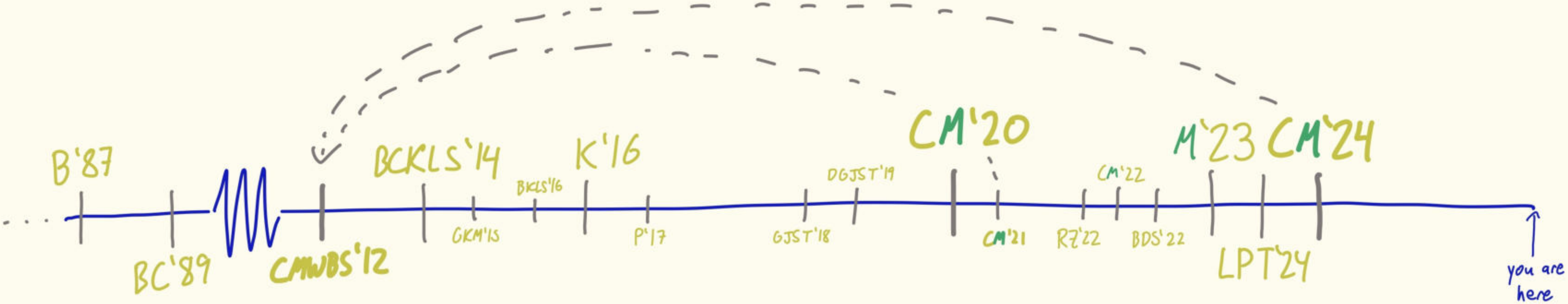


The History of Reuse



applications of catalytic ideas to ordinary space

The History of Reuse

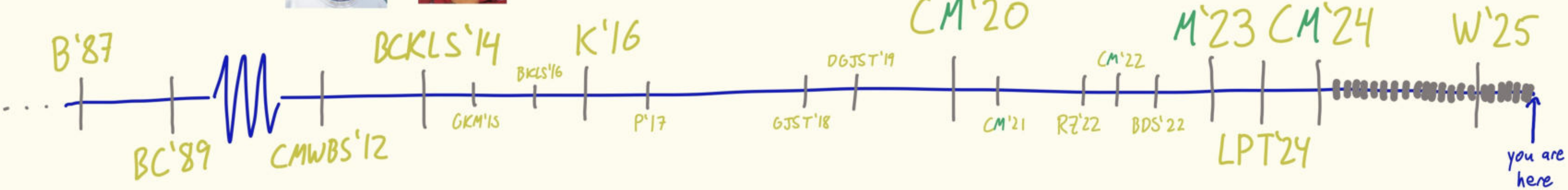


The History of Reuse

$\boxed{\text{SPACE}[t] \text{ requires } \text{TIME}[\gg t]}$

$\boxed{f^* \text{ in } \text{SPACE}[t \log t]}$

The History of Reuse



... and the field explodes

S'24
GJST'24
P'24
CLMP'25
KMPS'25
PSW'25

S'25
FMST'25
AFMSV'25
BFMSSST'25
AM'25
PT'25

AAV'25
DPTW'25
CGMPS'25
BDRS'25
CP'25
...

THE POWER OF REUSE

COR: compress-or-random
AR: arithmetic reversibility

functions doable with catalytic space
(which are unknown without it):

The Power of Reuse

COR: compress-or-random
AR: arithmetic reversibility

functions doable with catalytic space (which are unknown without it):

1. determinant[AR]

$$\det \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc$$

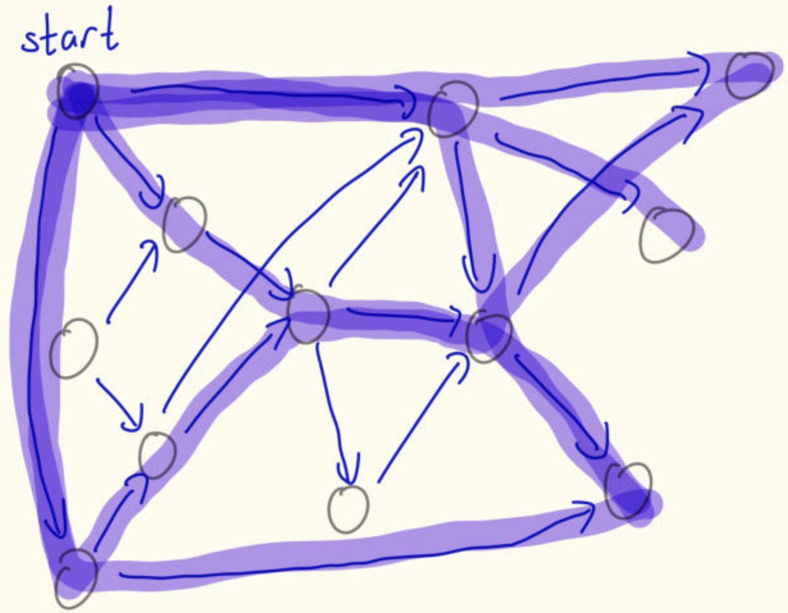
[BCKLS'14]

THE POWER OF REUSE

COR: compress-or-random
AR: arithmetic reversibility

functions doable with catalytic space (which are unknown without it):

1. determinant[AR]
2. random walks[COR]



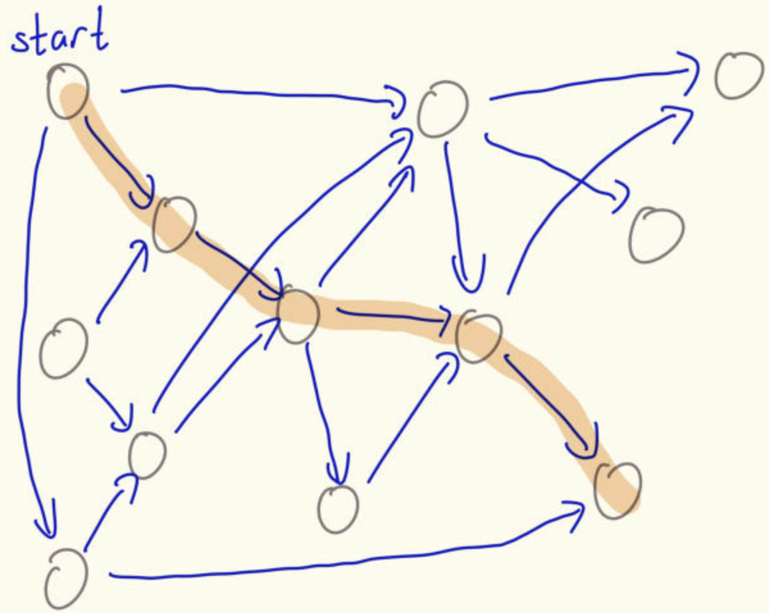
[D'15]

The Power of Reuse

COR: compress-or-random
AR: arithmetic reversibility

functions doable with catalytic space
(which are unknown without it):

1. determinant [AR]
2. random walks [COR]
3. connectivity [AR]



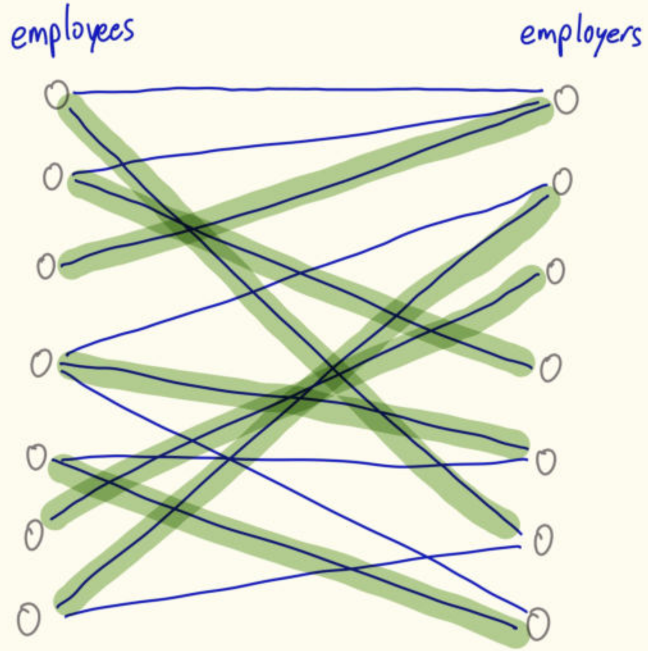
[BCKLS'14]

The Power of Reuse

COR: compress-or-random
AR: arithmetic reversibility

functions doable with catalytic space (which are unknown without it):

1. determinant AR
2. random walks COR
3. connectivity AR
4. matchings COR + AR



[AM'25]
[AAV'25]

THE POWER OF REUSE

COR: compress-or-random
AR: arithmetic reversibility

functions doable with catalytic space (which are unknown without it):

1. determinant[AR]
2. random walks[COR]
3. connectivity[AR]
4. matchings[COR + AR]

etc.

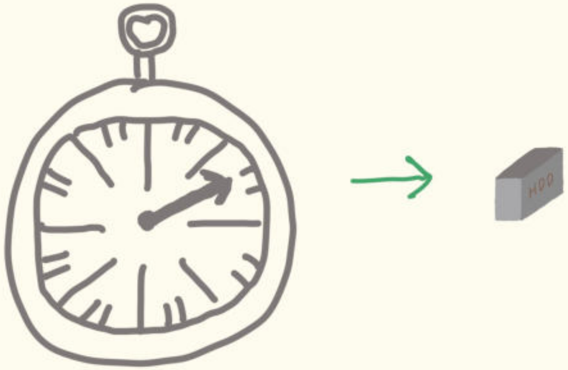
The Power of Reuse

implications of catalytic space

THE POWER OF REUSE

implications of catalytic space

1. SPACE vs TIME

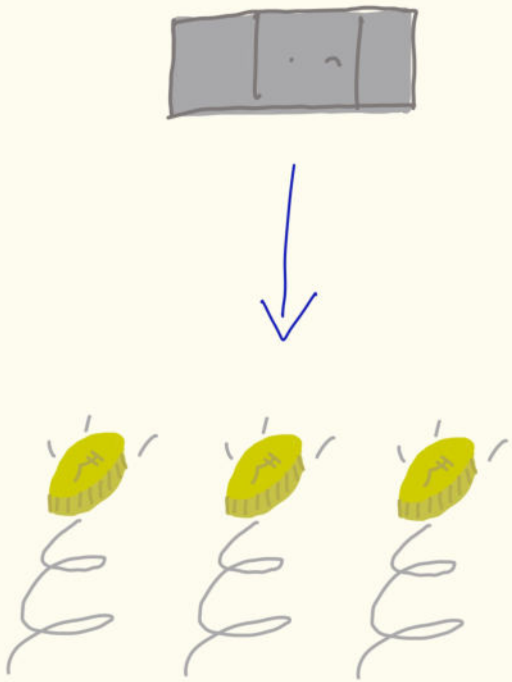


[W'25]

THE POWER OF REUSE

implications of catalytic space

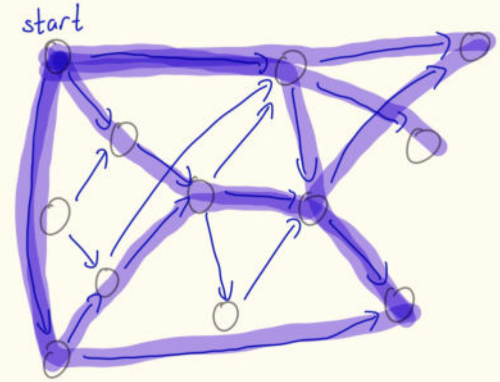
1. SPACE vs TIME
2. derandomization



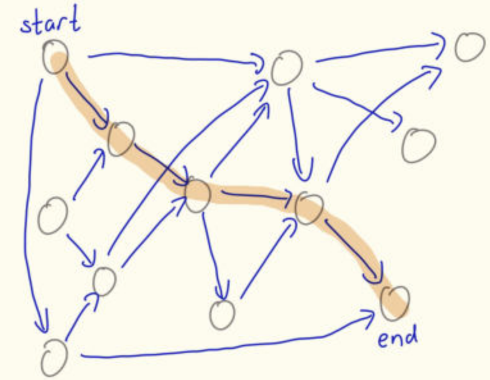
[DPT'24]
[LPT'24]
[CLMP'25]
[DPTW'25]

The Power of Reuse

implications of catalytic space



1. SPACE vs TIME
2. derandomization
3. simple walk algorithms

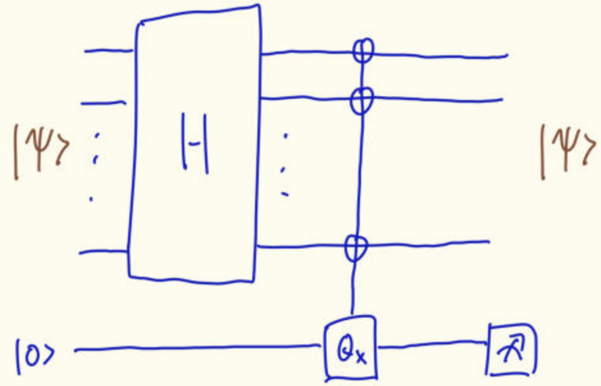


[KMPS'25]
[CP'25]

THE POWER OF REUSE

implications of catalytic space

1. SPACE vs TIME
2. derandomization
3. simple walk algorithms
4. quantum space



[BFMSSST'25]

[SMLBH'25]

THE POWER OF REUSE

implications of catalytic space

1. SPACE vs TIME
2. derandomization
3. simple walk algorithms
4. quantum space

etc.

Concluding Remarks

Summing Up

- memory can be used for both storage and computation at the same time

SUMMING UP

- memory can be used for both storage and computation at the same time

- catalytic computing has led to breakthroughs on the study of time and space

SUMMING UP

- memory can be used for both storage and computation at the same time

- catalytic computing has led to breakthroughs on the study of time and space

- the theory of computation still holds many surprises (so be careful assuming!)

THE FUTURE OF REUSE

1. Give a simple, direct proof of uSTConn $\in$ L.
2. Give a simple, direct proof of uSTConn $\in$ CL.
3. Give a simple, direct proof of STConn $\in$ CL.
4. Try to improve Savich's Theorem: prove $\mathsf{NSPACE}(s) \subseteq \mathsf{SPACE}(o(s^2))$.
5. Improve the deterministic space complexity of $\mathsf{BPSPACE}(s)$.
6. Decide the space complexity of TreeEval.
7. Give a register program for computing any polynomial $p(x_1 \ldots x_n)$ using $O(n)$ registers over a constant size ring $\mathcal{R}$ and $O(1)$ recursive calls to the input x.
8. Show that for any branching program B of sufficiently large width $w = \Omega(1)$ and length ℓ, there exists a branching program B' of width $w/2$ and length $O(\ell)$ computing the same function.
9. Show that for any branching program B of sufficiently large width w and length ℓ, there exists a branching program B' of width $w - 1$ and length poly (ℓ) computing the same function.
10. Find any function whose optimal space algorithm can be made almost entirely catalytic, i.e. a function requiring—or even that we only know how to do in—$\mathsf{SPACE}(s)$ but which is computable in $\mathsf{CSPACE}(\ll s, \approx s)$.
11. Prove $\mathsf{CL} \subseteq \mathsf{P}$.
12. Show that $\mathsf{P} \not\subseteq \mathsf{L/poly}$ implies $\mathsf{CL} \subseteq \mathsf{P}$.
13. Show that $\mathsf{CL} \subseteq \mathsf{P}$ would give strong evidence $\mathsf{ZPP} \subseteq \mathsf{P}$.
14. Show that NC^2, or even any circuit of $\omega(\log n)$ depth, can be computed in CL.
15. Give a register program for computing x^k in the non-commutative setting using linear space and a constant number of recursive calls to x.
16. Show that $\mathsf{BPNC}^1 \subseteq \mathsf{CL}$.
17. Design a catalytic branching program with $2^{O(n)}$ start nodes and total size $2^{O(n)} \cdot O(n)$ for any function f.
18. What is the power of $\mathsf{CL}/poly$, and does it have a natural syntactic characterization?
19. Show the existence of an oracle D such that $\mathsf{CL}^D = \mathsf{EXP}^D$.
20. Extend the $\mathsf{BPL} \subseteq \mathsf{CL}$ simulation to show $\mathsf{CBPL} \subseteq \mathsf{CL}$.
21. Show that CL is equivalent even if we allow $\omega(1)$ many errors on the catalytic tape at the end, or alternatively if we allow $O(1)$ such errors in expectation over all inputs x and catalytic tapes τ.
22. Utilize non-determinism in conjunction with catalytic computing in a non-trivial way.
23. Prove $\mathsf{CNSPACE}(s, c) \subseteq \mathsf{CSPACE}(s^2, c^2)$.
24. Implement a catalytic algorithm such that it is actually useful.
25. What does quantum catalytic space look like?
26. Devise a register program using basic instructions inspired by unitary computation, and use it to show non-trivial results for e.g. BQP.
27. Devise a circuit that uses known results from space reuse and catalytic computing to efficiently solve some problem in a way that we do not know how to do directly.
28. Show $\mathsf{TC}^1 \subseteq \mathsf{VP}$.
29. Is the network coding conjecture true or false?
30. Prove or disprove the network coding conjecture when all nodes are restricted to sending linear transformations of their incoming messages.
31. Is there a meaningful notion of a catalytic data structure, or is there anything to be gained from a data structure stored in catalytic memory?
32. Show CL is contained in some subclass of P, perhaps NC, given a believable cryptographic assumption.
33. Show evidence against objects in cryptography based on techniques in reusing space.
34. Show the existence, conditional or otherwise, of a natural class of cryptographic objects by using clean computation.
35. Prove that the existence of one-way functions in CL, or even any one-way function computable by a poly-size poly-length register program, implies the existence of one-way functions in NC^0.

source: [Mer`23]

The Future of Reuse

1. Give a simple, direct proof of uSTConn $\in$ L.
2. Give a simple, direct proof of uSTConn $\in$ CL.
3. Give a simple, direct proof of STConn $\in$ CL.
4. Try to improve Savich's Theorem: prove NSPACE$(s) \subseteq$ SPACE$(o(s^2))$.
5. Improve the deterministic space complexity of BPSPACE(s).
6. Decide the space complexity of TreeEval.
7. Give a register program for computing any polynomial $p(x_1 \dots x_n)$ using $O(n)$ registers over a constant size ring $\mathcal{R}$ and $O(1)$ recursive calls to the input x.
8. Show that for any branching program B of sufficiently large width $w = \Omega(1)$ and length ℓ, there exists a branching program B' of width $w/2$ and length $O(\ell)$ computing the same function.
9. Show that for any branching program B of sufficiently large width w and length ℓ, there exists a branching program B' of width $w - 1$ and length poly (ℓ) computing the same function.
10. Find any function whose optimal space algorithm can be made almost entirely catalytic, i.e. a function requiring—or even that we only know how to do in—SPACE(s) but which is computable in CSPACE$(\ll s, \approx s)$.
11. Prove CL $\subseteq$ P.
12. Show that P $\not\subseteq$ L/poly implies CL $\subseteq$ P.
13. Show that CL $\subseteq$ P would give strong evidence ZPP $\subseteq$ P.
14. Show that NC^2, or even any circuit of $\omega(\log n)$ depth, can be computed in CL.
15. Give a register program for computing x^k in the non-commutative setting using linear space and a constant number of recursive calls to x.
16. Show that BPNC$^1 \subseteq$ CL.
17. Design a catalytic branching program with $2^{O(n)}$ start nodes and total size $2^{O(n)} \cdot O(n)$ for any function f.
18. What is the power of CL$/poly$, and does it have a natural syntactic characterization?
19. Show the existence of an oracle D such that CL^D = EXP^D.
20. Extend the BPL $\subseteq$ CL simulation to show CBPL $\subseteq$ CL.
21. Show that CL is equivalent even if we allow $\omega(1)$ many errors on the catalytic tape at the end, or alternatively if we allow $O(1)$ such errors in expectation over all inputs x and catalytic tapes τ.
22. Utilize non-determinism in conjunction with catalytic computing in a non-trivial way.
23. Prove CNSPACE$(s, c) \subseteq$ CSPACE(s^2, c^2).
24. Implement a catalytic algorithm such that it is actually useful.
25. What does quantum catalytic space look like?
26. Devise a register program using basic instructions inspired by unitary computation, and use it to show non-trivial results for e.g. BQP.
27. Devise a circuit that uses known results from space reuse and catalytic computing to efficiently solve some problem in a way that we do not know how to do directly.
28. Show TC$^1 \subseteq$ VP.
29. Is the network coding conjecture true or false?
30. Prove or disprove the network coding conjecture when all nodes are restricted to sending linear transformations of their incoming messages.
31. Is there a meaningful notion of a catalytic data structure, or is there anything to be gained from a data structure stored in catalytic memory?
32. Show CL is contained in some subclass of P, perhaps NC, given a believable cryptographic assumption.
33. Show evidence against objects in cryptography based on techniques in reusing space.
34. Show the existence, conditional or otherwise, of a natural class of cryptographic objects by using clean computation.
35. Prove that the existence of one-way functions in CL, or even any one-way function computable by a poly-size poly-length register program, implies the existence of one-way functions in NC^0.

source: [Mer'23]

Resources

Reusing Space: Techniques and Open Problems

Ian Mertz*

Abstract

In the world of space-bounded complexity, there is a strain of results showing that space can, somewhat paradoxically, be used for multiple purposes at once. Touchstone results include Barrington's Theorem and the recent line of work on catalytic computing. We refer to such techniques, in contrast to the usual notion of reclaiming space, as *reusing space*.

In this survey we will dip our toes into the world of reusing space. We do so in part by studying techniques, viewed through the lens of a few highlight results, but our main focus will be the wide variety of open problems in the field.

In addition to the broader and more challenging questions, we aim to provide a number of questions that are fairly simple to state, have clear practical and theoretical implications, and, most importantly, that a newcomer with little background experience can still sit down and play with for a while.

Catalytic computation

Michal Koucký*
Computer Science Institute
Charles University, Prague
koucky@iuuk.mff.cuni.cz

Abstract

Catalytic computation was defined by Buhrman et al. (STOC, 2014). It addresses the question whether memory, that already stores some unknown data that should be preserved for later use, can be meaningfully used for computation. Buhrman et al. provide an intriguing answer to this question by giving examples where the occupied memory can be used to perform computation. In this expository article we survey what is known about this problem and how it relates to other problems.

Resources

iuuk.mff.cuni.cz/~iwmertz/

introduction [my main interests] | previous work [past activities] | methodology [teaching & students]
main results [publications] | acknowledgements [funding, positions, & visits] | appendix [the fun stuff]

catalytic computation & reusing space

How useful is *full memory* as a computational resource? Imagine trying to solve some functions on a computer with only limited memory, but now you are also given additional access to a *massive hard drive* which it can freely use, provided it *keeps all the initial data on the hard drive intact* at the end of its computation. Considering this data could be arbitrary—obviously this memory has nothing to do with the problem at hand—does this hard drive give us *any additional power*?

The surprising answer is that this hard drive, which we call *catalytic memory*, is very powerful. First, it gives us at least as much power as any other well-studied resource, be it randomness or non-determinism; in fact, *catalytic memory alone* is as powerful as being given *catalytic memory, randomness, and non-determinism simultaneously*. Second, the techniques and subroutines developed for this catalytic computation model, which I (uncreatively) refer to as *reusing space*, have given *major breakthrough results in the ordinary space-bounded setting*, mostly notably Williams' recent simulation of time t in space $\sqrt{t \log t}$.

My central goal is to understand and characterize this catalytic model, as well as to further use the techniques developed therein to solve longstanding open questions about space.

[survey (EATCS)] [techniques] **(more resources coming soon)**

techniques (catalytic computing)

Here is an overview of some of the major arguments/techniques that appear in the catalytic literature. If you're here I'm assuming you know the setup for the model; if not then check out a survey article such as mine or Michal Koucky's, or even the original paper by Buhrman et al. (a great read!) to get oriented. For more info on specifics, I would suggest checking out the resources page for suggested papers, talks, etc.

Take a click on whatever strikes your fancy!

compress-or-random

register programs

structure in catalytic space

introduction

The only trivial upper bound on catalytic space is an equal amount of pure space. In order to improve this result, we need to look deeper into the structure of catalytic algorithms. We will borrow from two fundamental results on ordinary space: first, the straightforward fact that space s algorithms can be simulated in *time* $\exp(s)$; and second, the much more intriguing (and much more recently discovered) fact that all algorithms can be made *reversible* with only a constant amount of extra space.

average case time

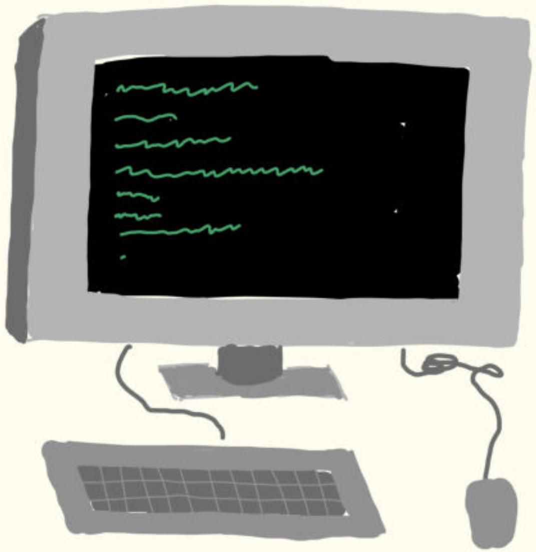
open problems database to come...

memory

runtime

error correction

logic



circuitry

learning

parallelism

randomness


That's all Folks!