TEL AVIV UNIVERSITY

Optimization When You Don’t Know the Future

Roie Levin

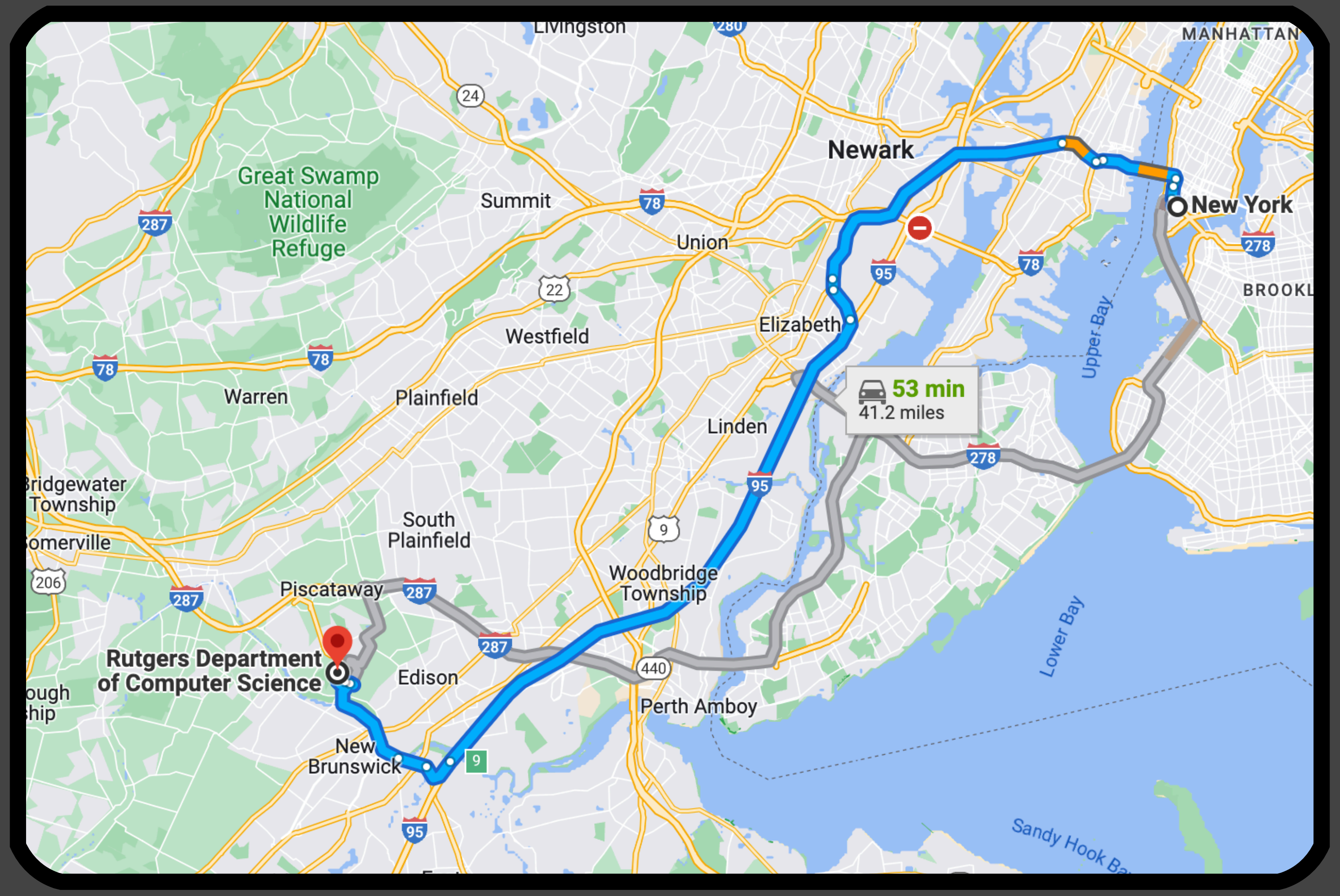
Introduction

My Research

I research algorithms
for optimization
in the face of uncertainty.

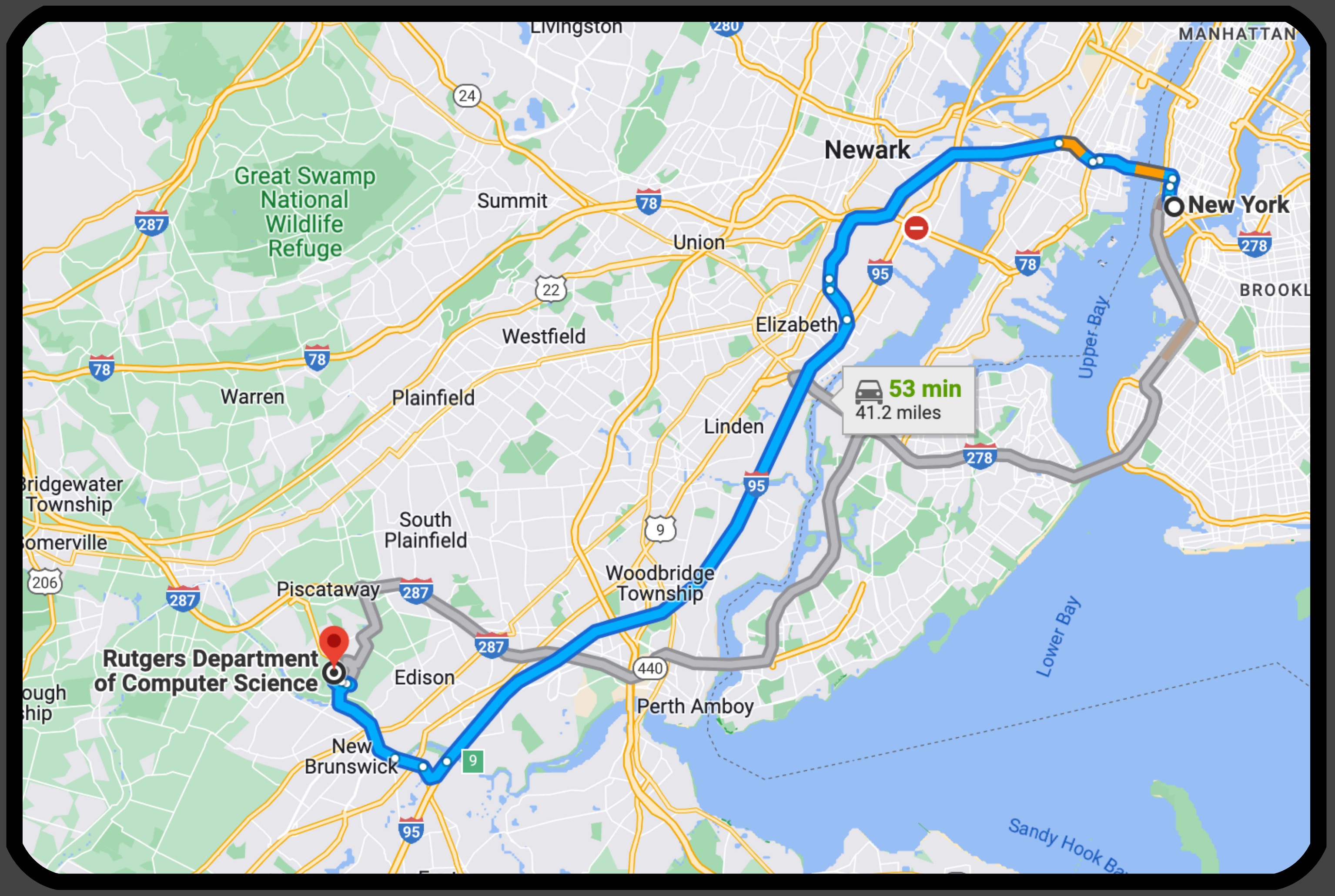
Classical CS is about Computational Challenges

Classical CS is about Computational Challenges



Classical CS is about Computational Challenges

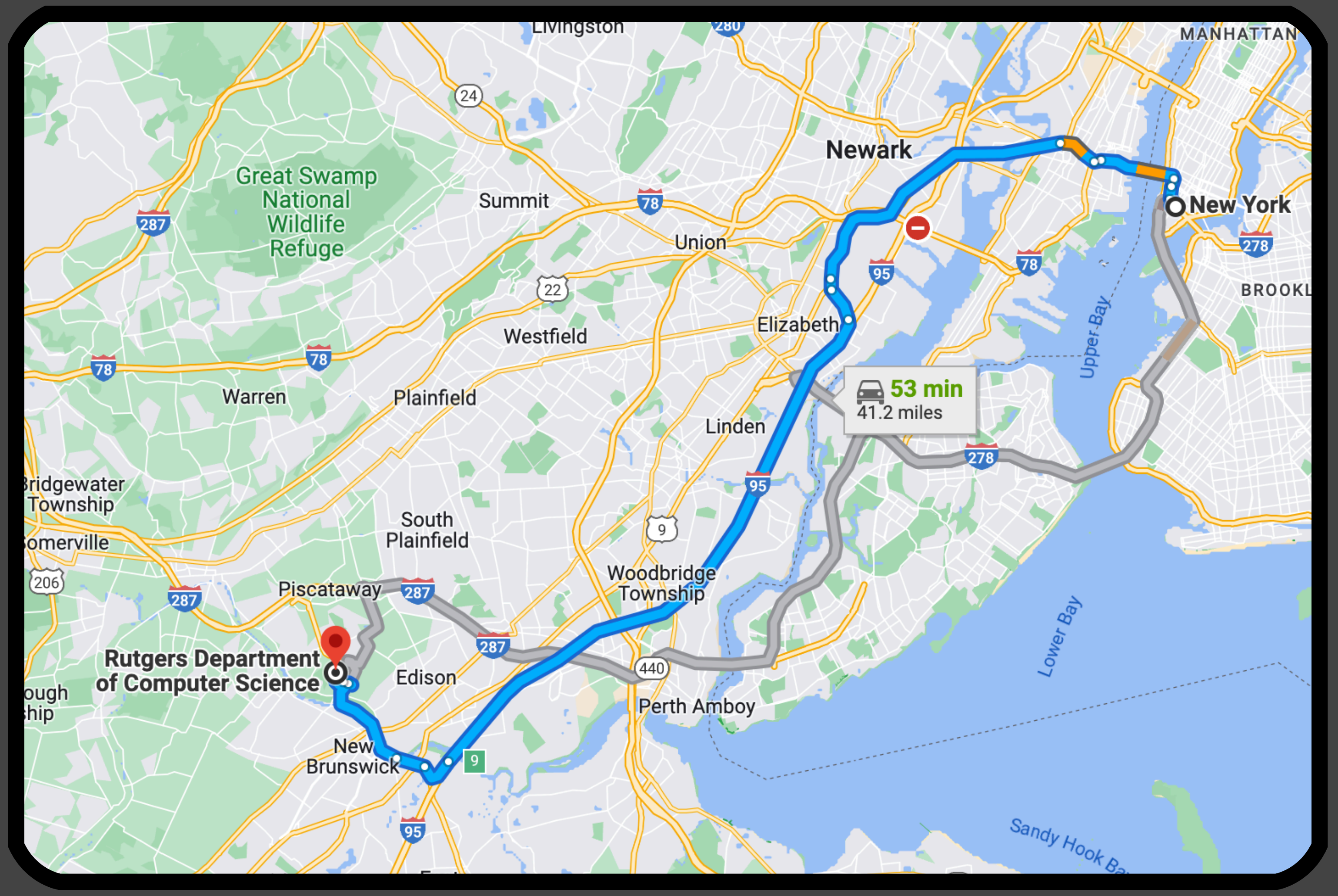
`ShortestPath`



`Knapsack`

Classical CS is about Computational Challenges

`ShortestPath`

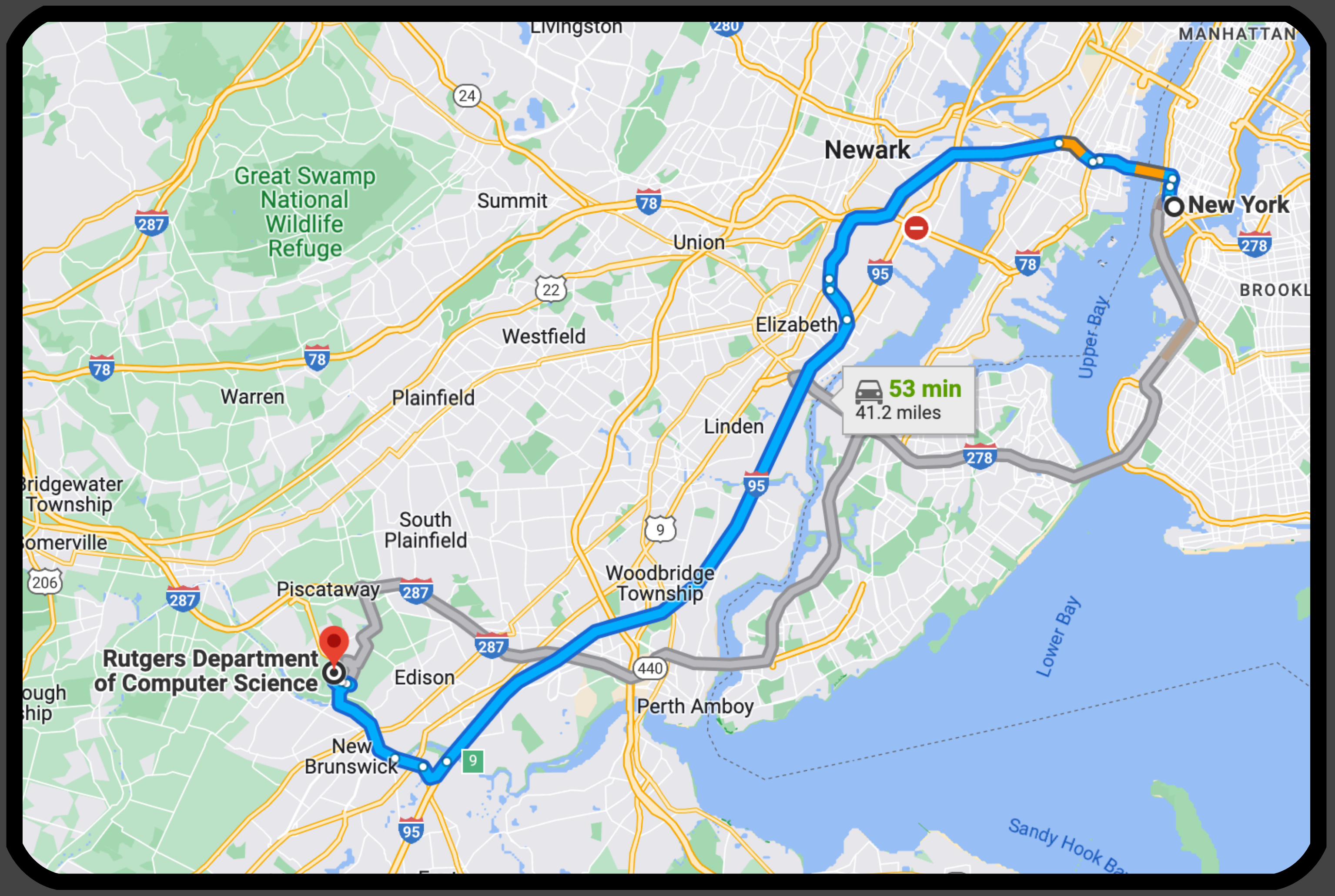


`Knapsack`

Computationally Easy → Computationally Hard

Classical CS is about Computational Challenges

`ShortestPath`



`Knapsack`

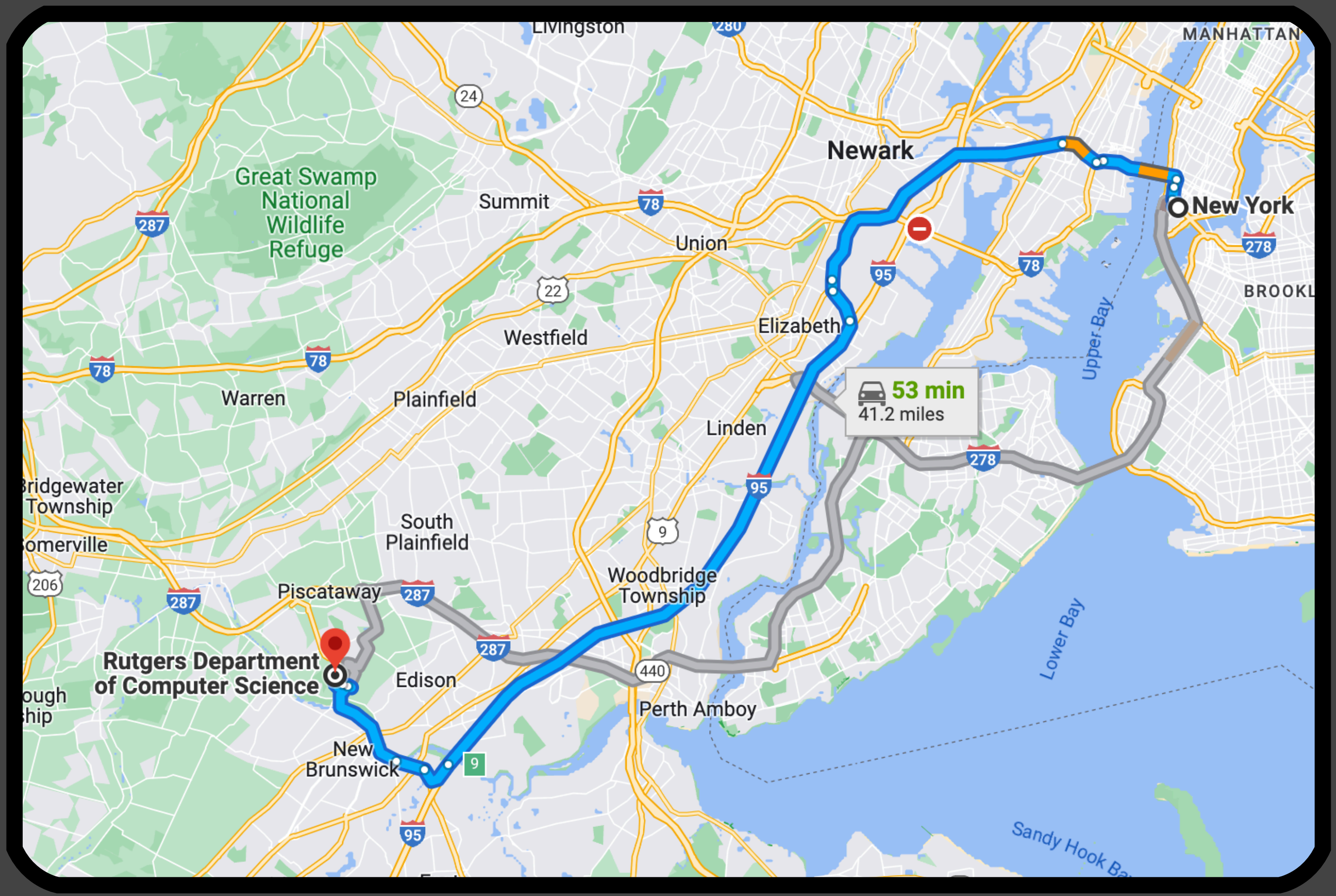
NP-hard

Computationally Easy

Computationally Hard

Classical CS is about Computational Challenges

ShortestPath



Approximate
Knapsack

Knapsack

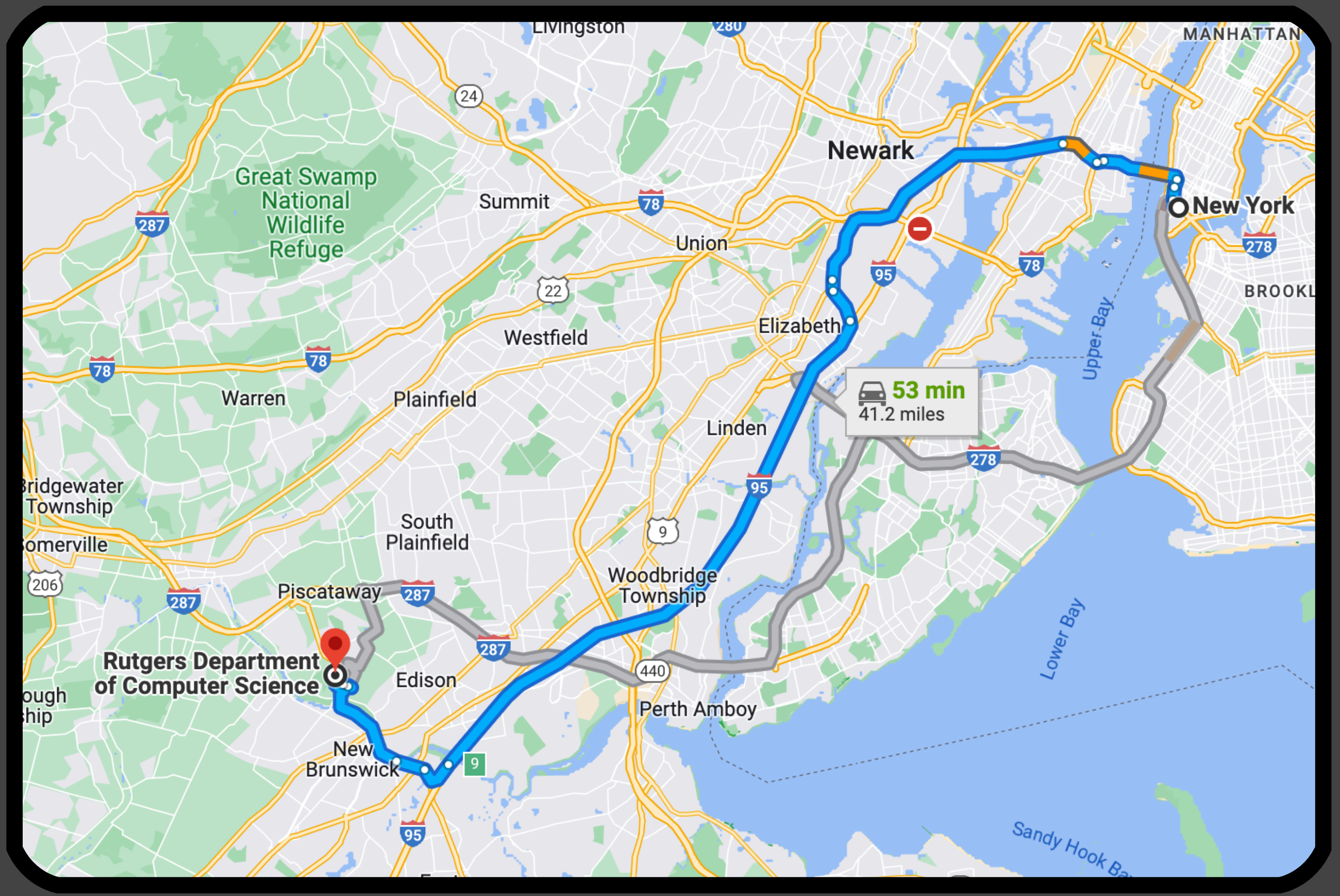
NP-hard

Computationally Easy

Computationally Hard

Classical CS is about Computational Challenges

`ShortestPath`



`Approximate Knapsack`

`Knapsack`

NP-hard

Computationally Easy → Computationally Hard

Beautiful theory of Approximation Algorithms!

A Different Source of Hardness: Uncertainty

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4

A Different Source of Hardness: Uncertainty

`FindMax`

4 1 10 -2 22 7

`Online FindMax`

4 1

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4	1	10

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4	1	10	-2

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4	1	10	-2	22

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4	1	10	-2	22	7

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4	1

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4	1	10

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4	1	10

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4	1	10	-2

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4	1	10	-2	22

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4	1	10	-2	22	7

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4	1	10	-2	22	7

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Online FindMax`

4	1	10	-2	22	7

Full Information → Uncertain

A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

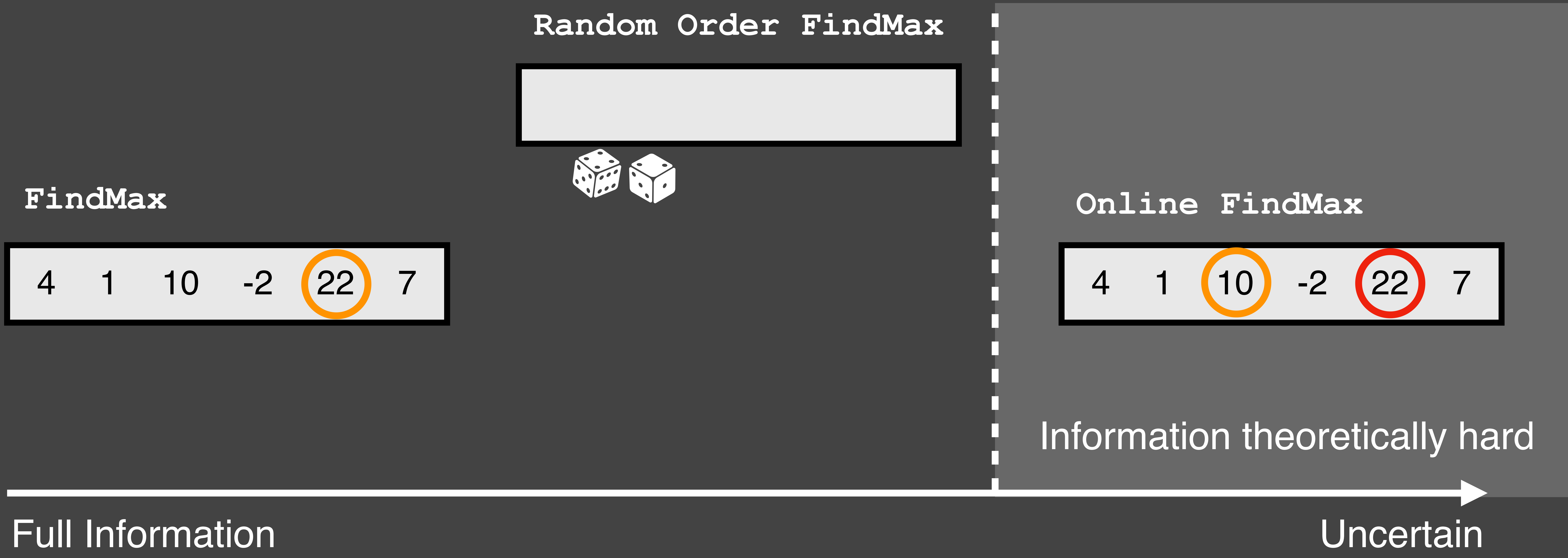
`Online FindMax`

4	1	10	-2	22	7

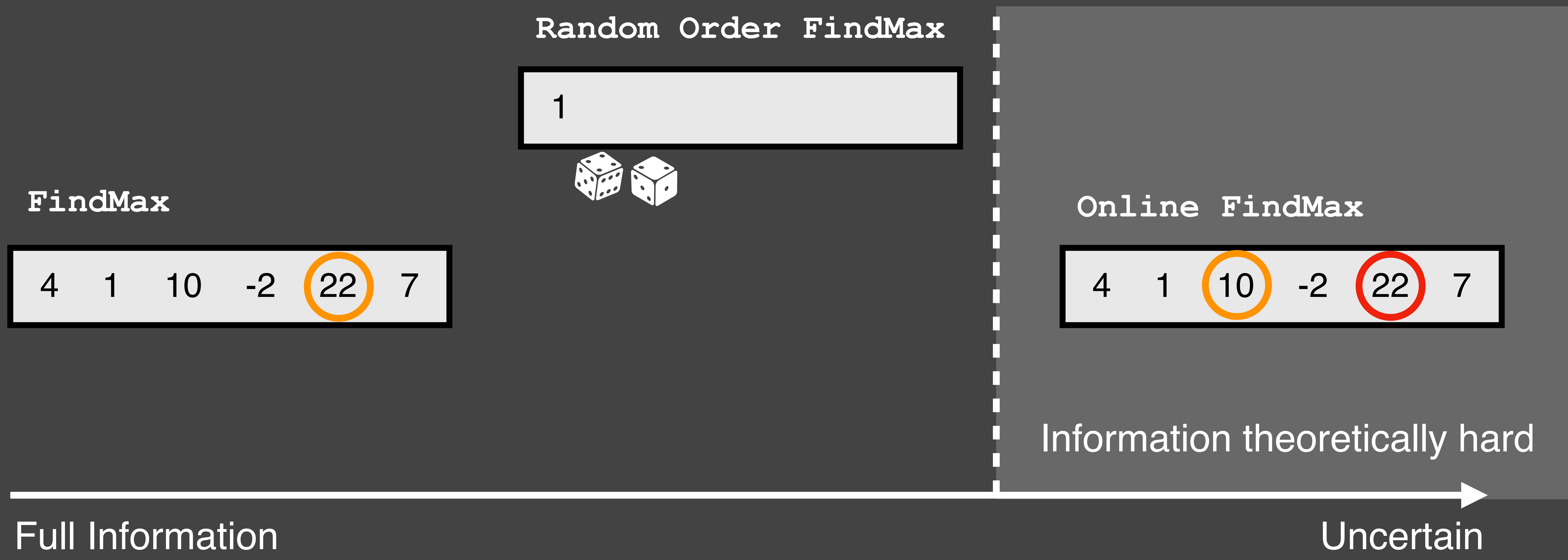
Information theoretically hard

Full Information → Uncertain

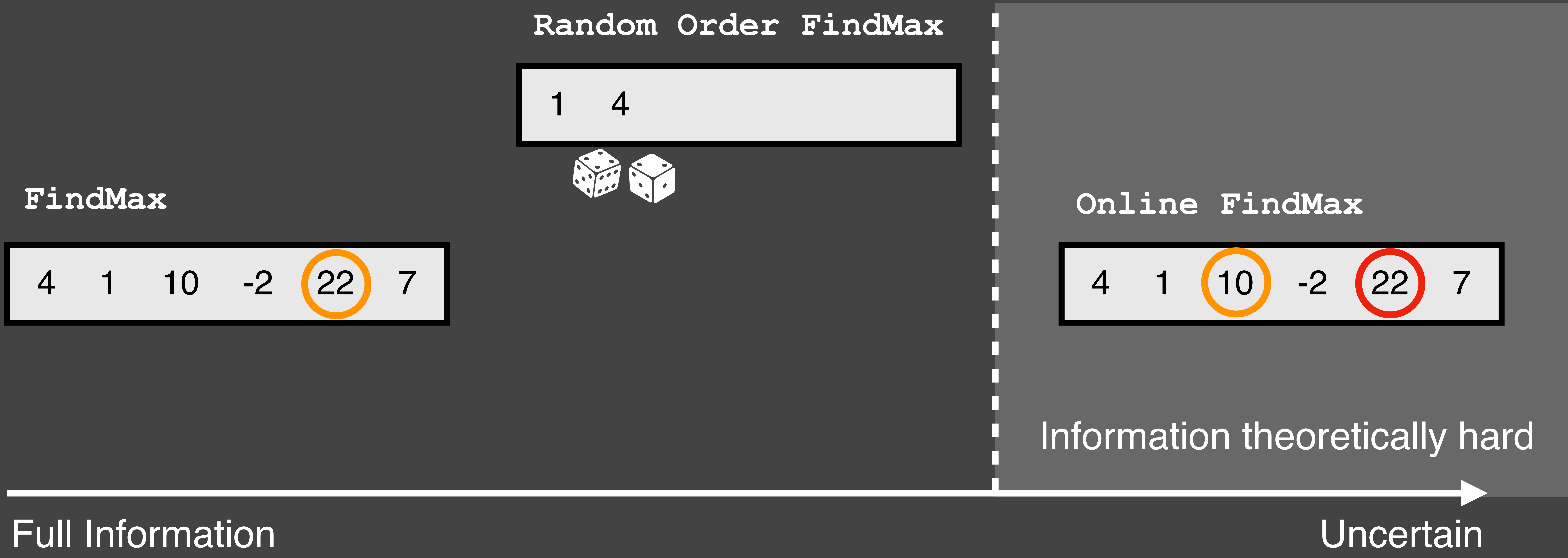
A Different Source of Hardness: Uncertainty



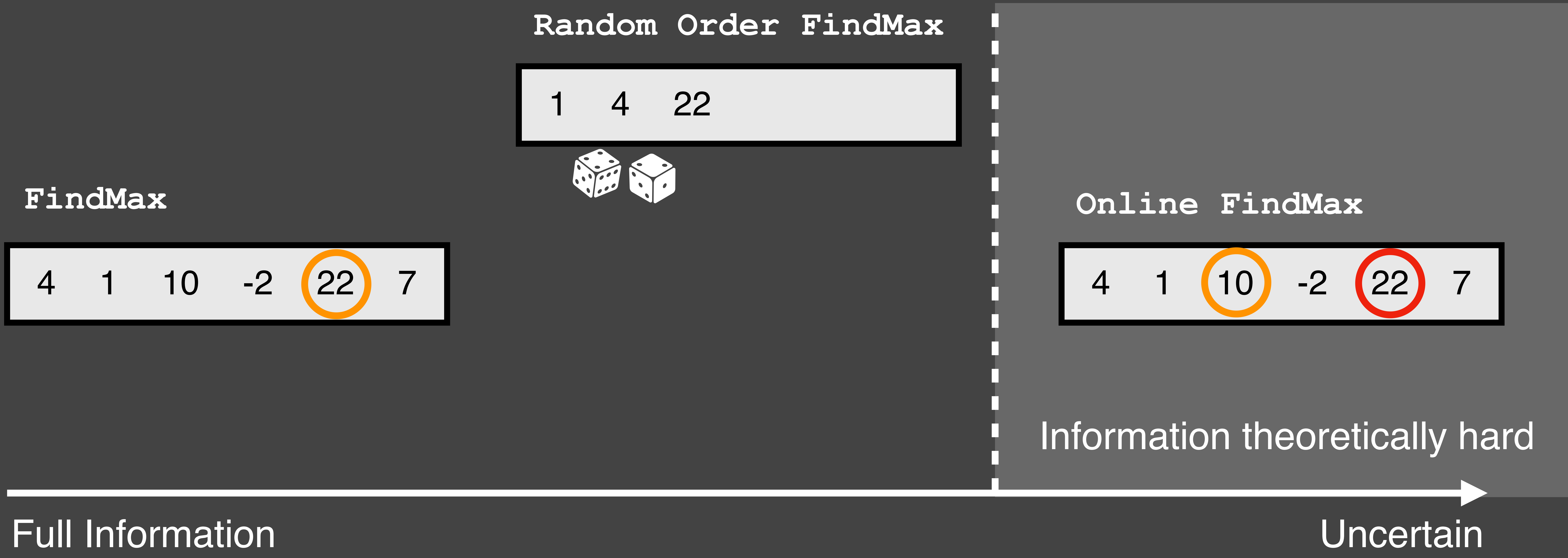
A Different Source of Hardness: Uncertainty



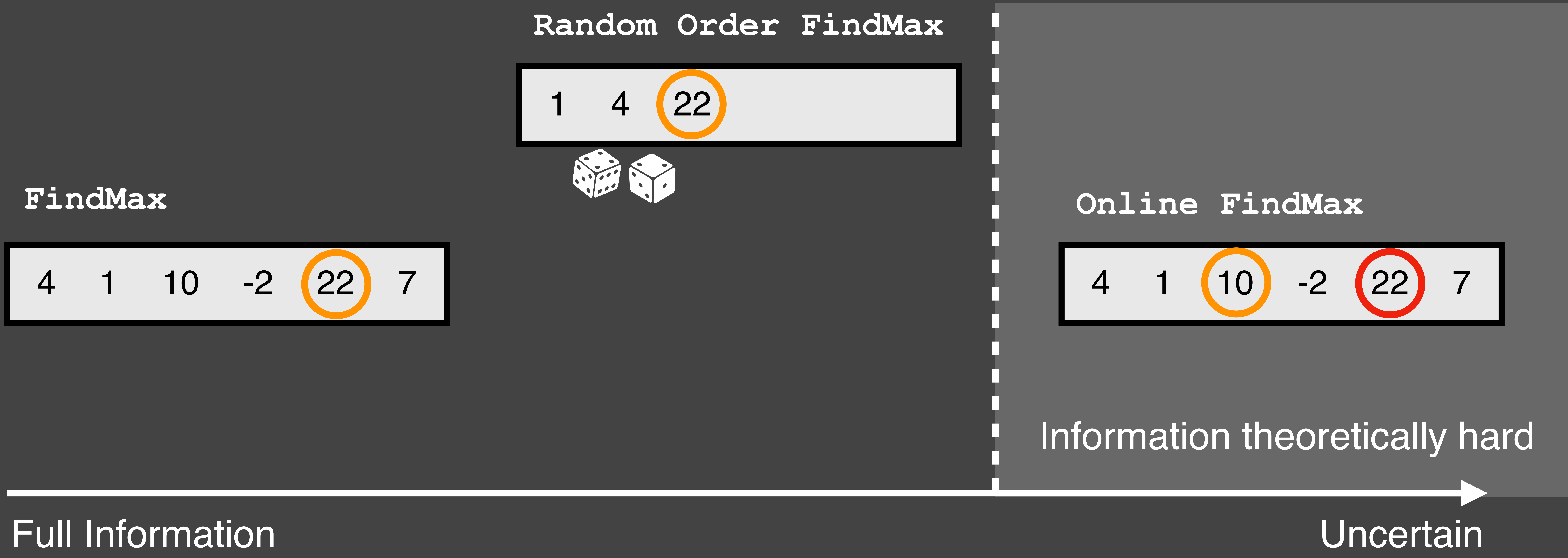
A Different Source of Hardness: Uncertainty



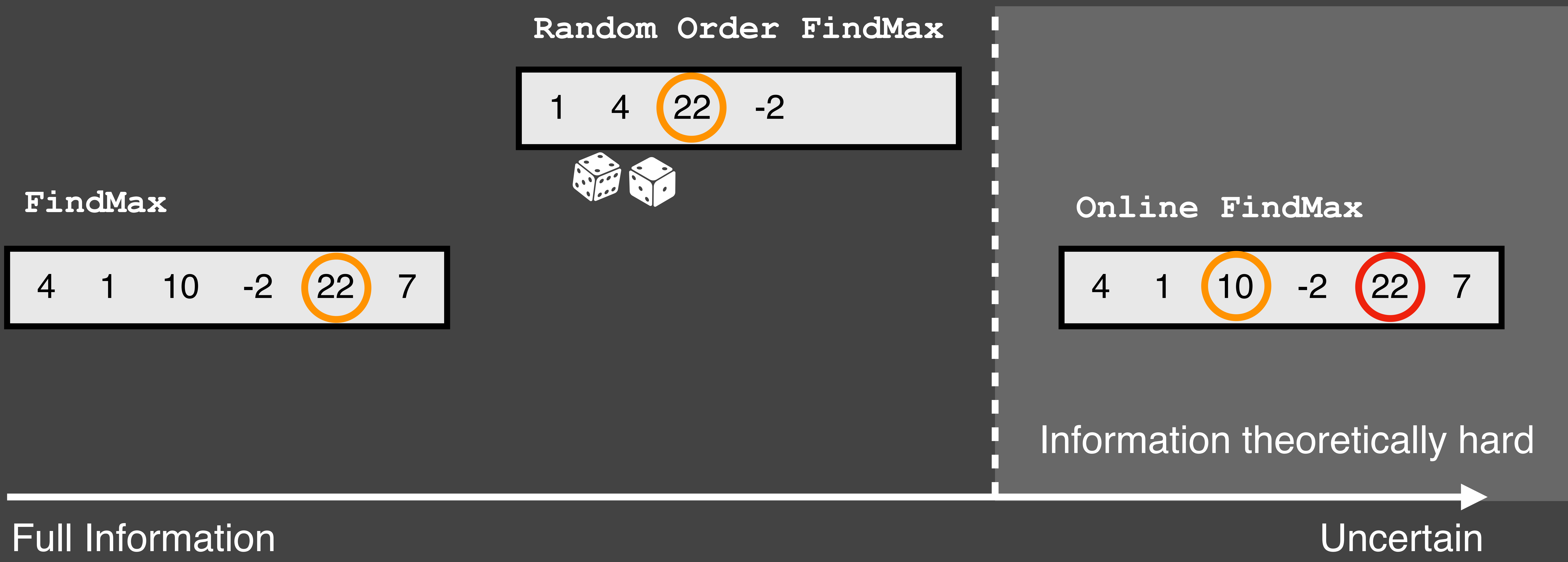
A Different Source of Hardness: Uncertainty



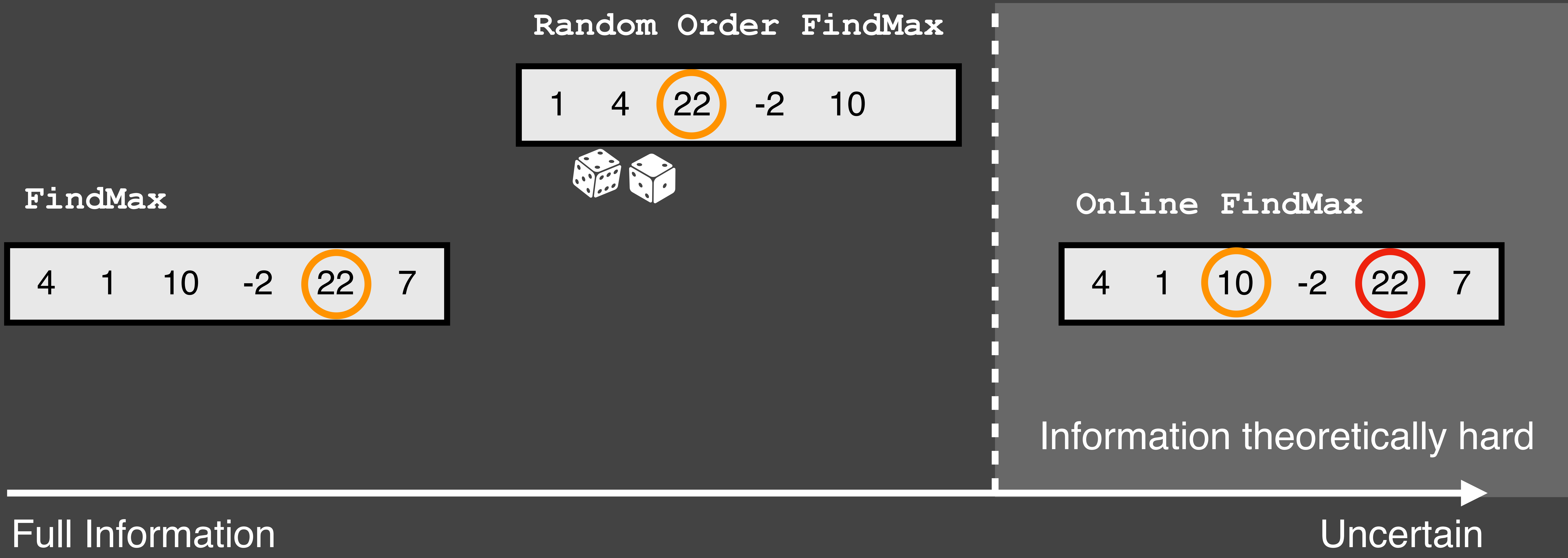
A Different Source of Hardness: Uncertainty



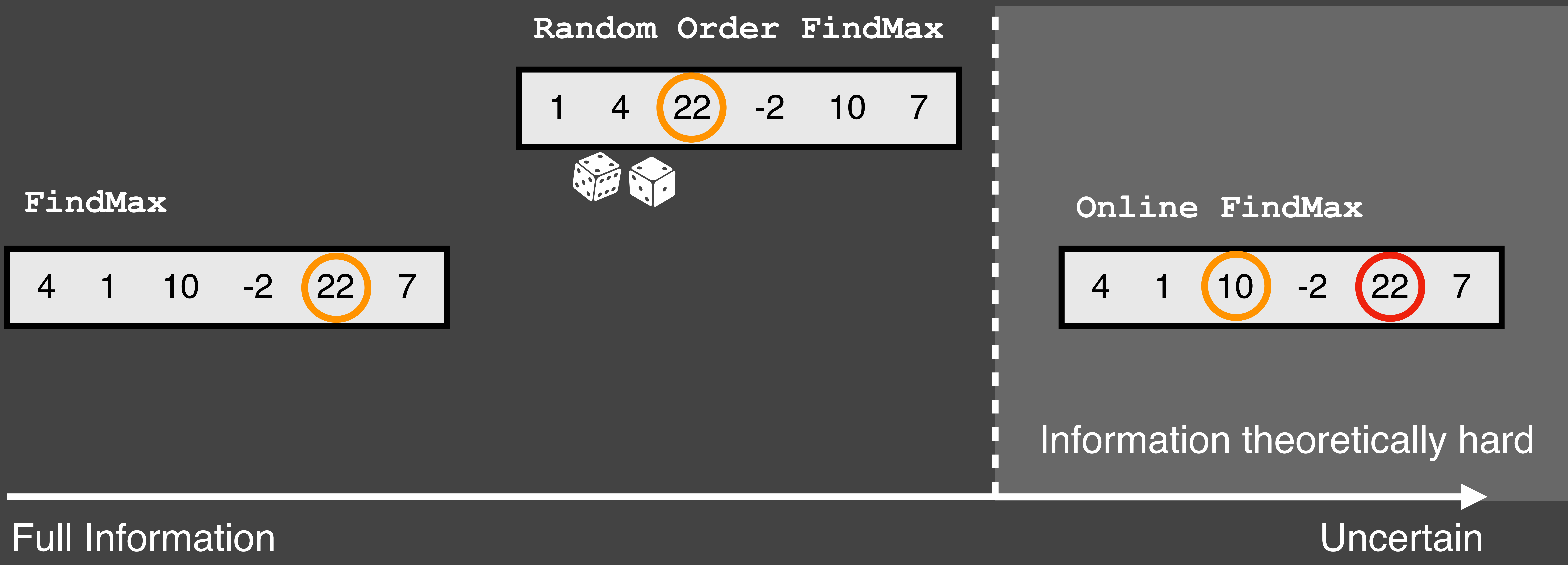
A Different Source of Hardness: Uncertainty



A Different Source of Hardness: Uncertainty



A Different Source of Hardness: Uncertainty



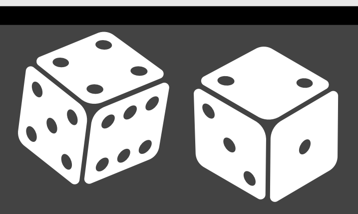
A Different Source of Hardness: Uncertainty

FindMax

4	1	10	-2	22	7

Random Order FindMax

1	4	22	-2	10	7



a.k.a. Secretary Problem

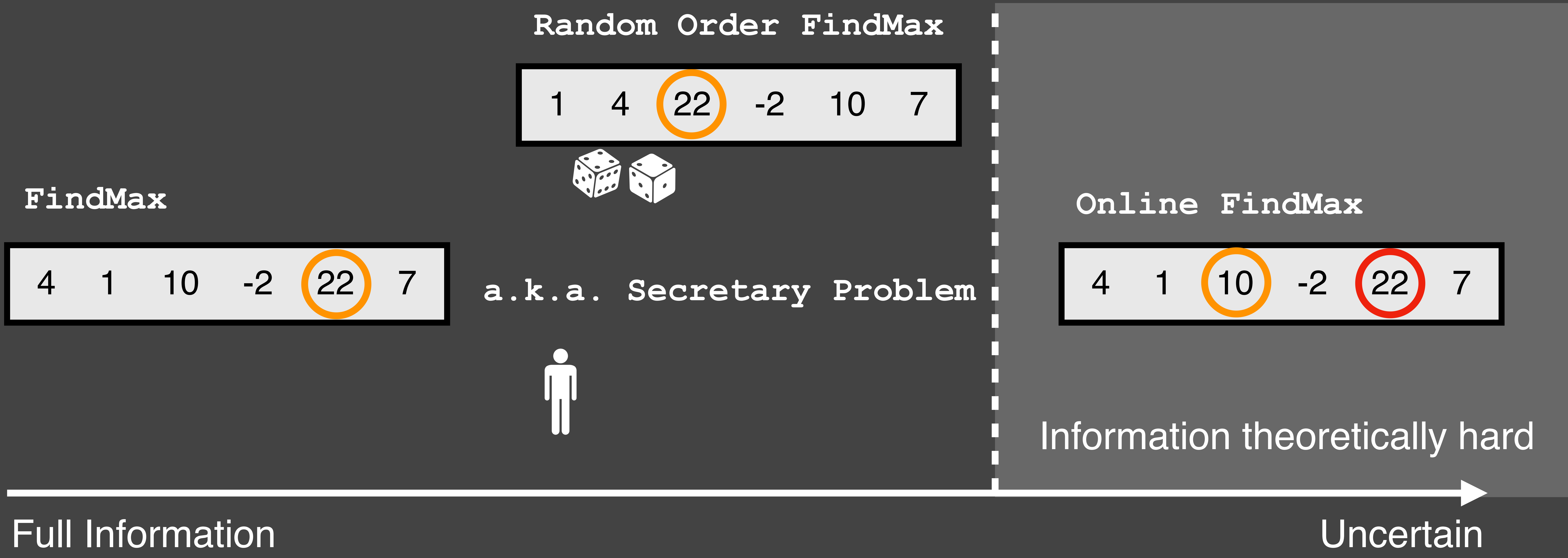
Online FindMax

4	1	10	-2	22	7

Information theoretically hard

Full Information → Uncertain

A Different Source of Hardness: Uncertainty



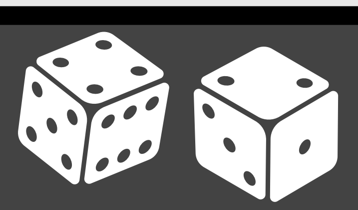
A Different Source of Hardness: Uncertainty

`FindMax`

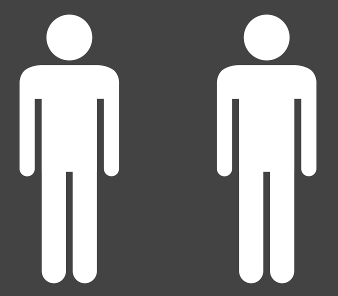
4	1	10	-2	22	7

`Random Order FindMax`

1	4	22	-2	10	7



`a.k.a. Secretary Problem`



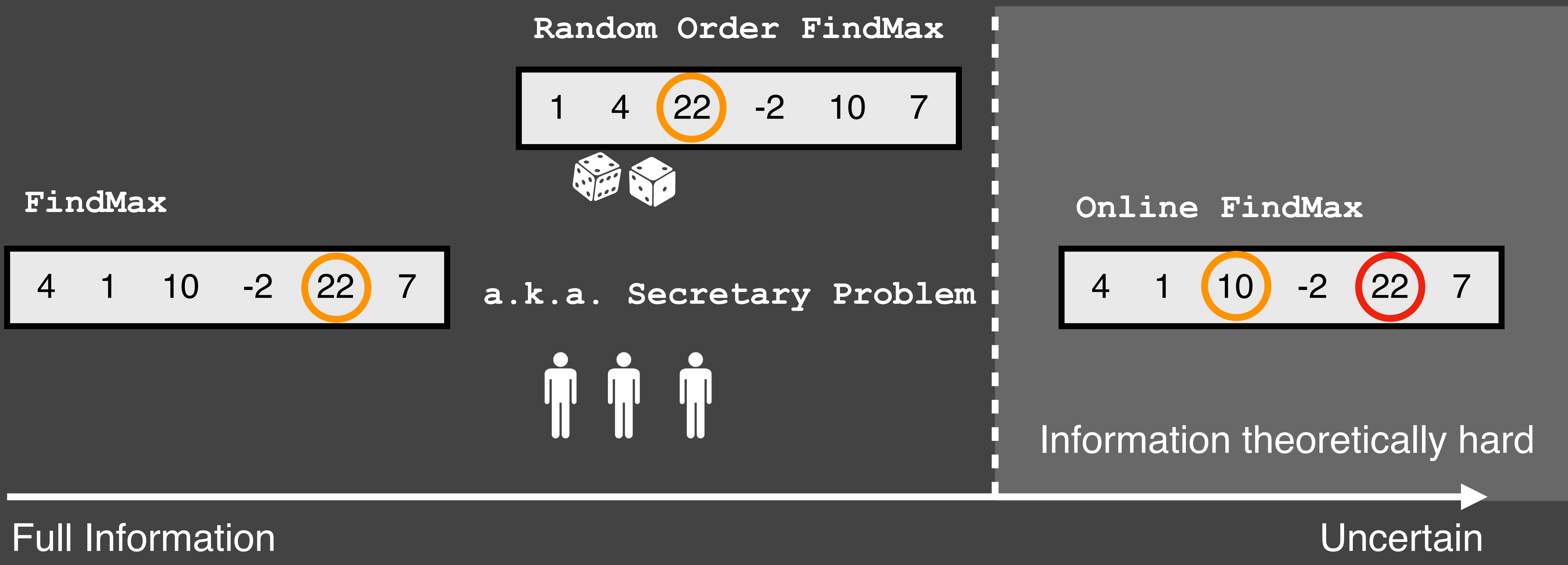
`Online FindMax`

4	1	10	-2	22	7

Information theoretically hard

Full Information → Uncertain

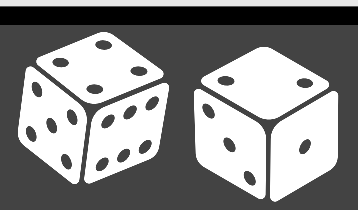
A Different Source of Hardness: Uncertainty



A Different Source of Hardness: Uncertainty

Random Order FindMax

1	4	22	-2	10	7



FindMax

4	1	10	-2	22	7

a.k.a. Secretary Problem

Online FindMax

4	1	10	-2	22	7

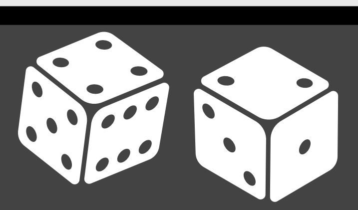
Information theoretically hard

Full Information → Uncertain

A Different Source of Hardness: Uncertainty

Random Order FindMax

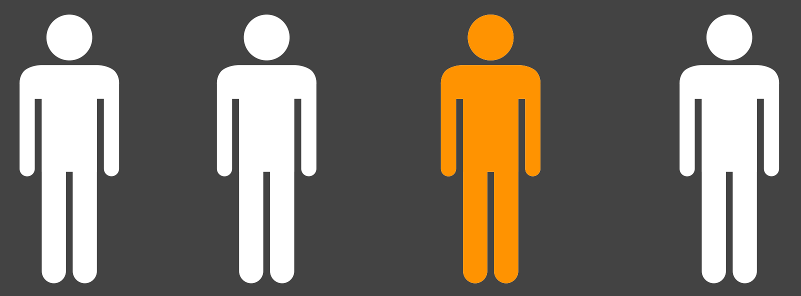
1	4	22	-2	10	7



FindMax

4	1	10	-2	22	7

a.k.a. Secretary Problem



Online FindMax

4	1	10	-2	22	7

Information theoretically hard

Full Information → Uncertain

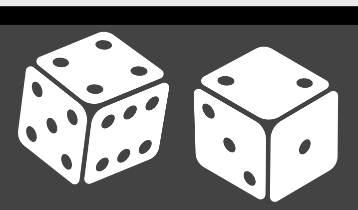
A Different Source of Hardness: Uncertainty

`FindMax`

4	1	10	-2	22	7

`Random Order FindMax`

1	4	22	-2	10	7



`a.k.a. Secretary Problem`

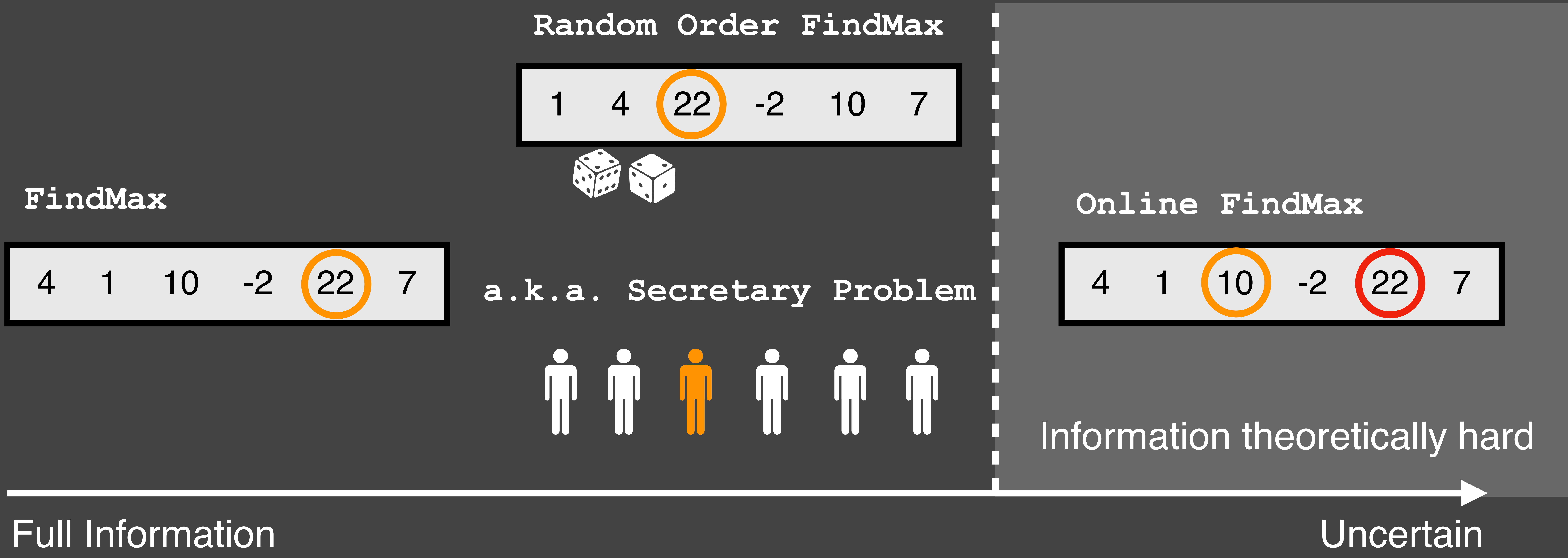
`Online FindMax`

4	1	10	-2	22	7

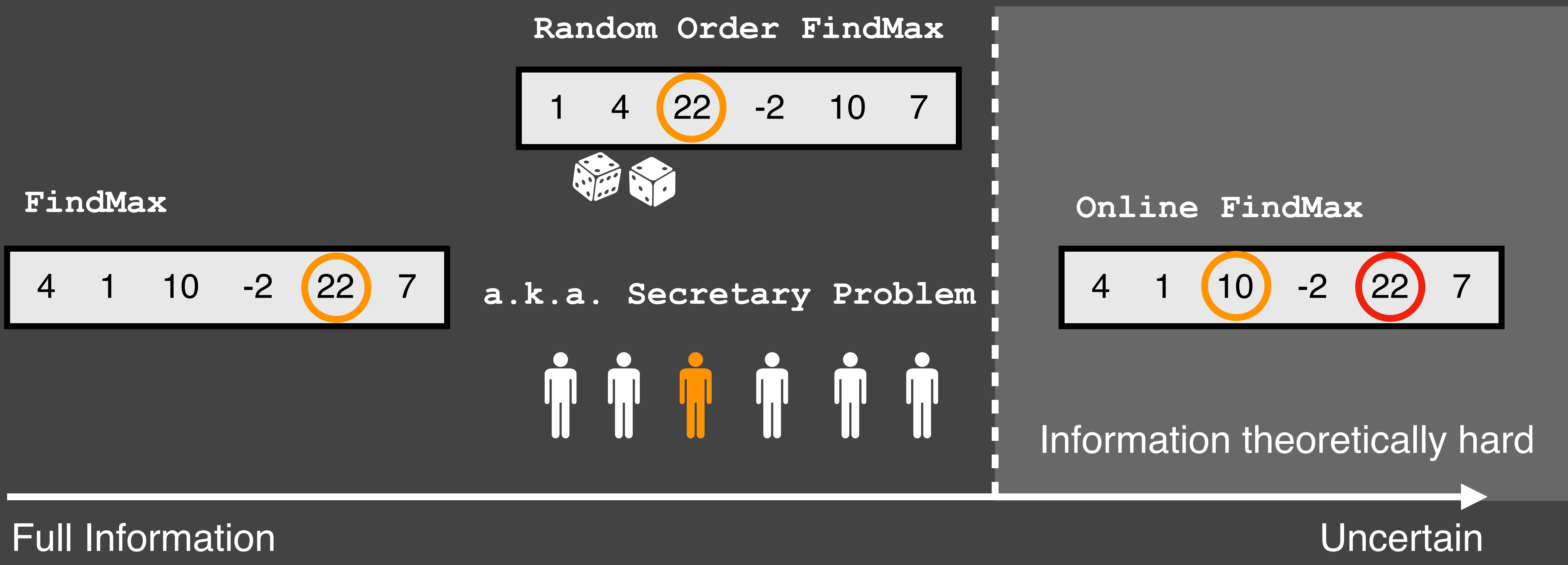
Information theoretically hard

Full Information → Uncertain

A Different Source of Hardness: Uncertainty



A Different Source of Hardness: Uncertainty

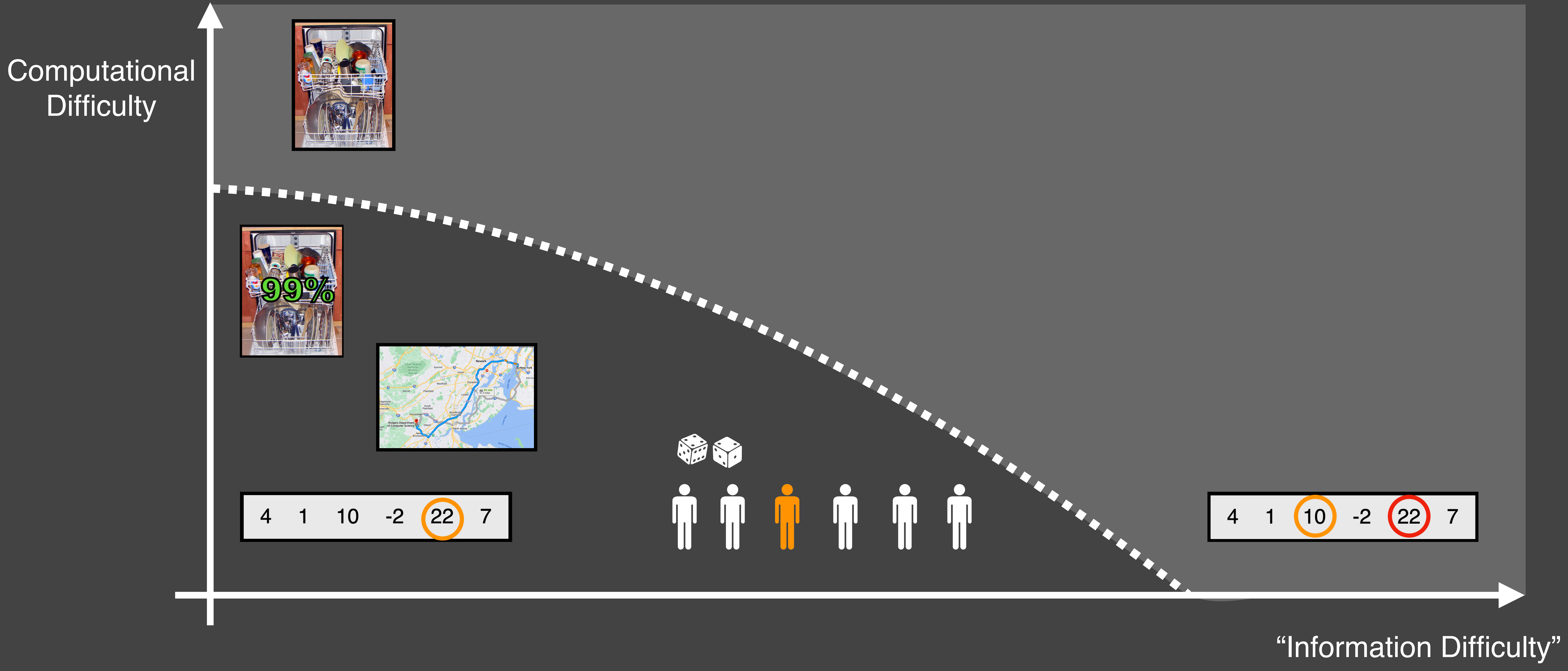


Beautiful theory of Decision Making Under Uncertainty!

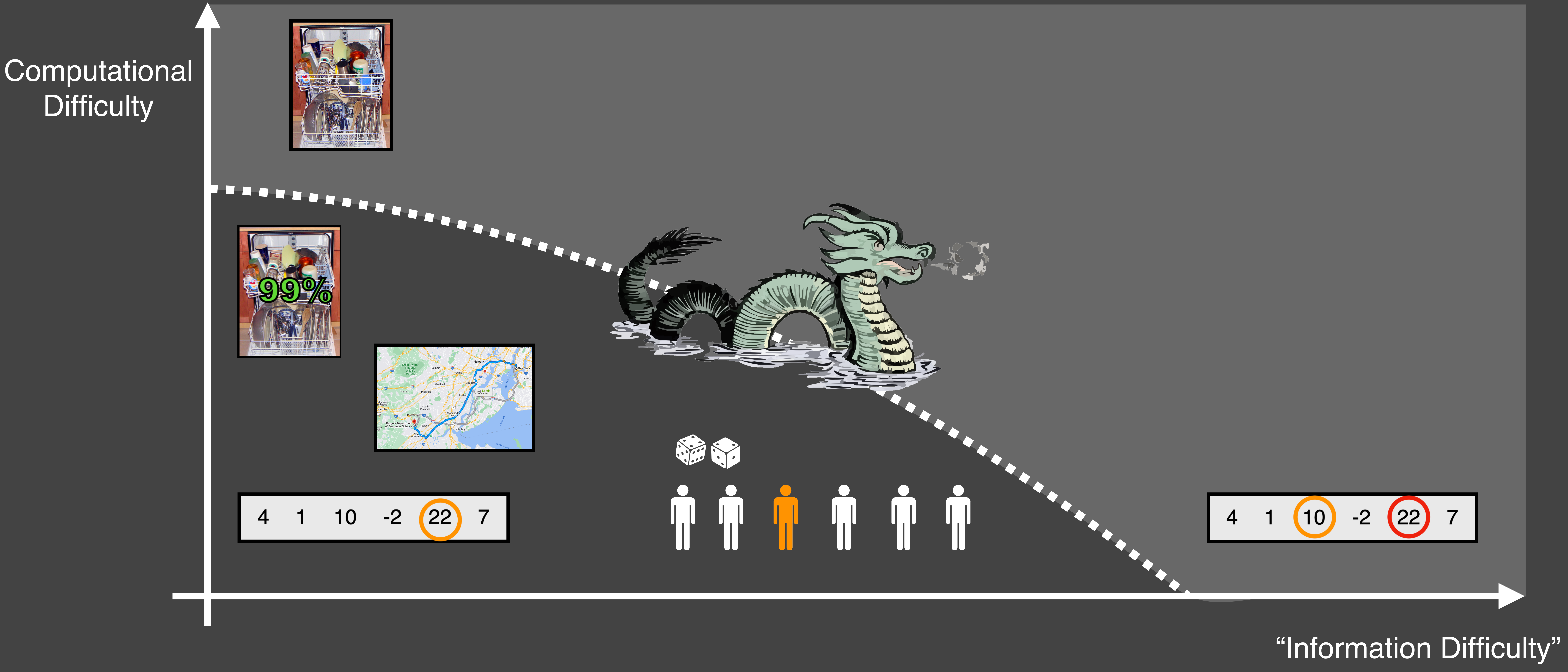
The Computation/Information Landscape

The Computation/Information Landscape

The Computation/Information Landscape



The Computation/Information Landscape



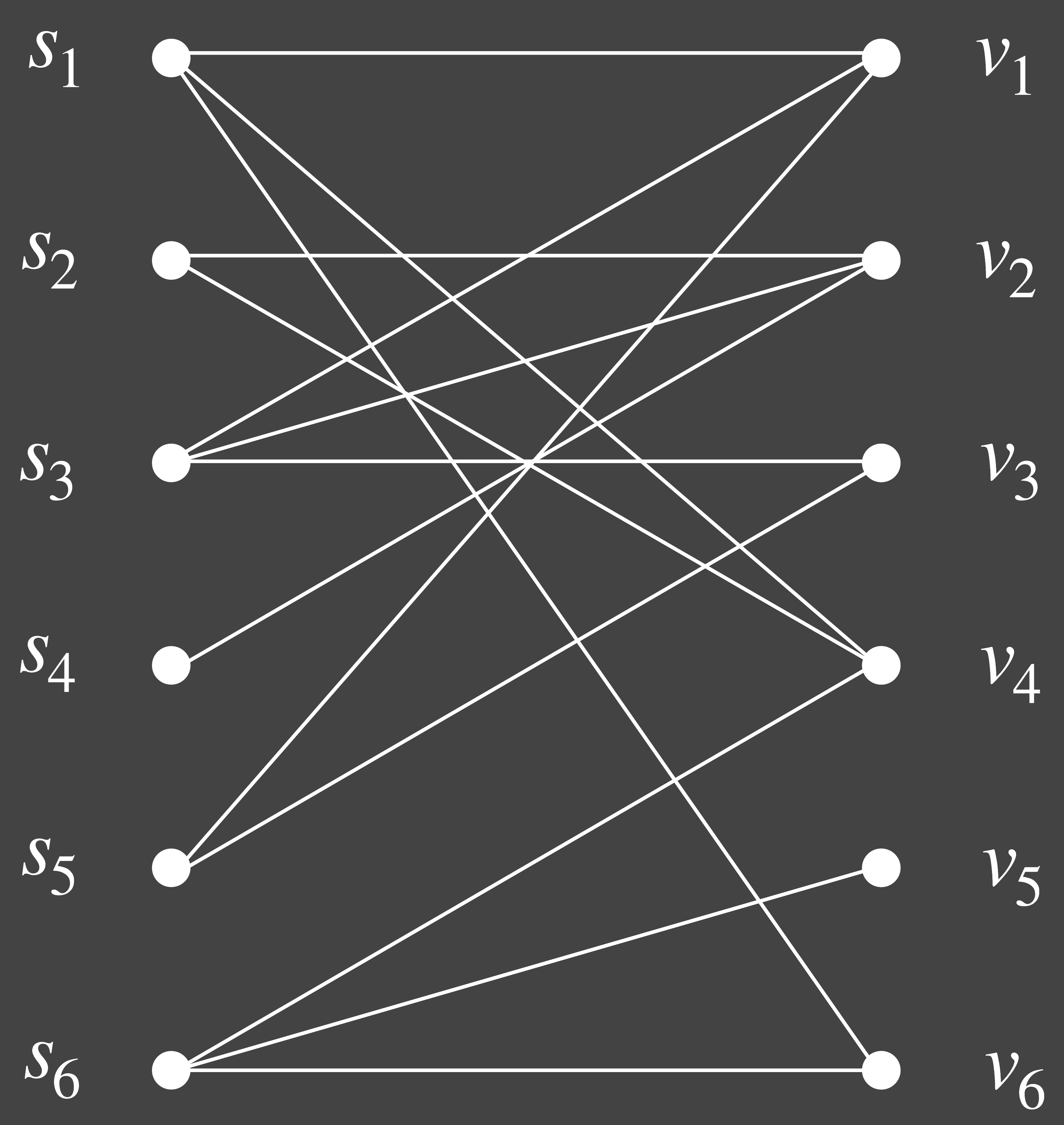
The Computation/Information Landscape

The Computation/Information Landscape

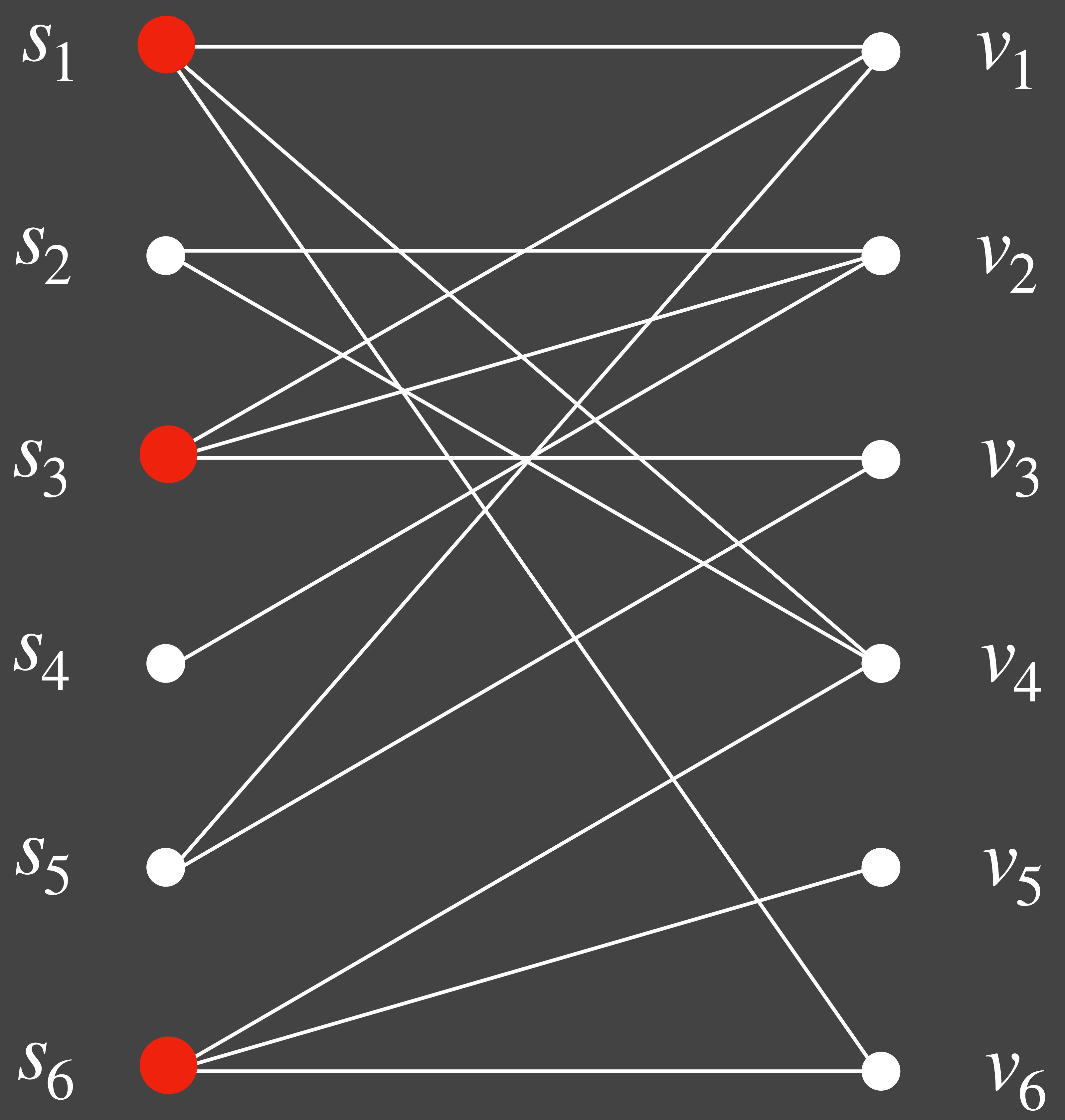
Q: What are the fundamental tradeoffs between computational resources and information?

My focus: approximation algorithms ∩ decision making under uncertainty.

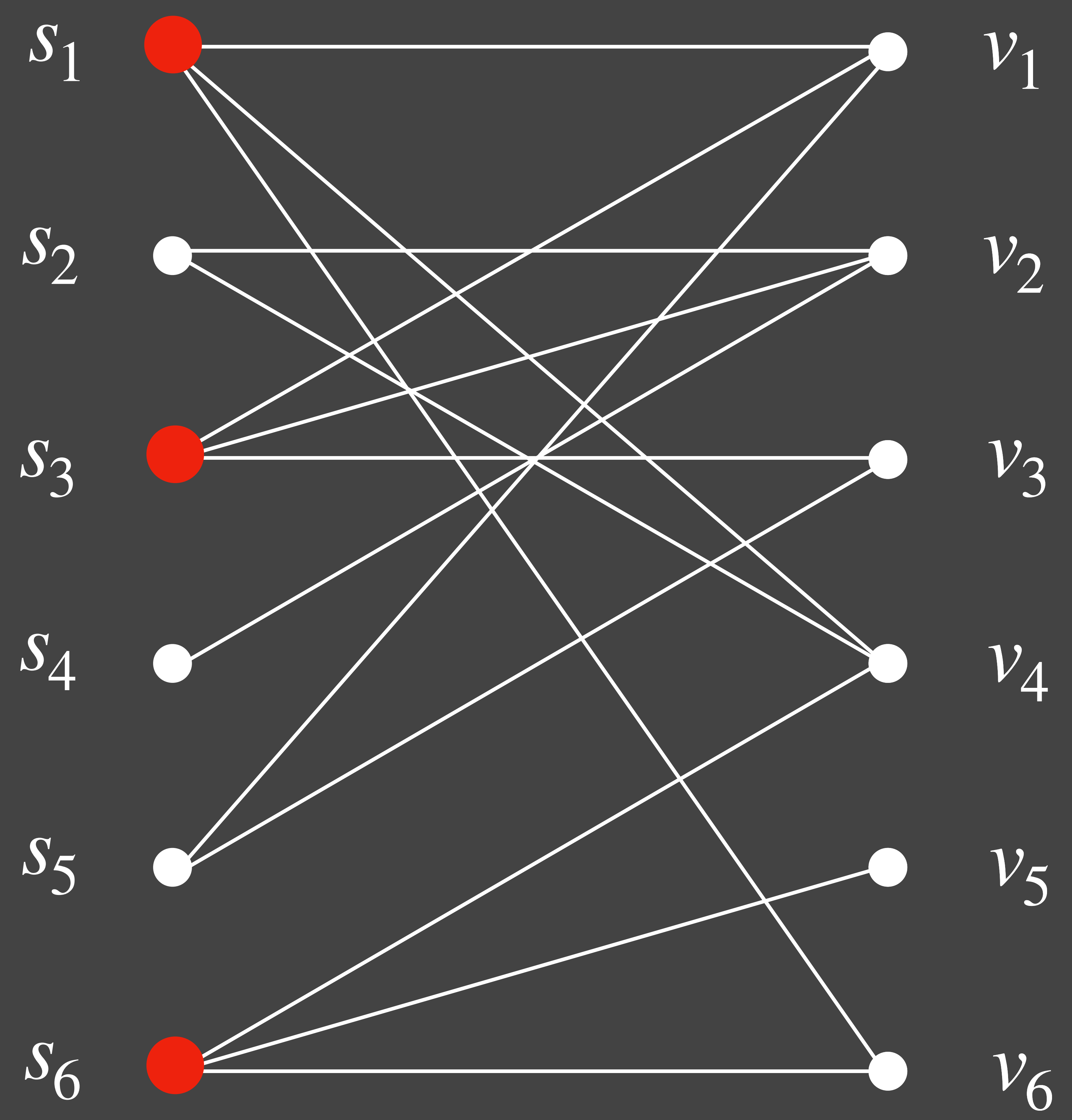
Running Example: Set Cover



Running Example: Set Cover

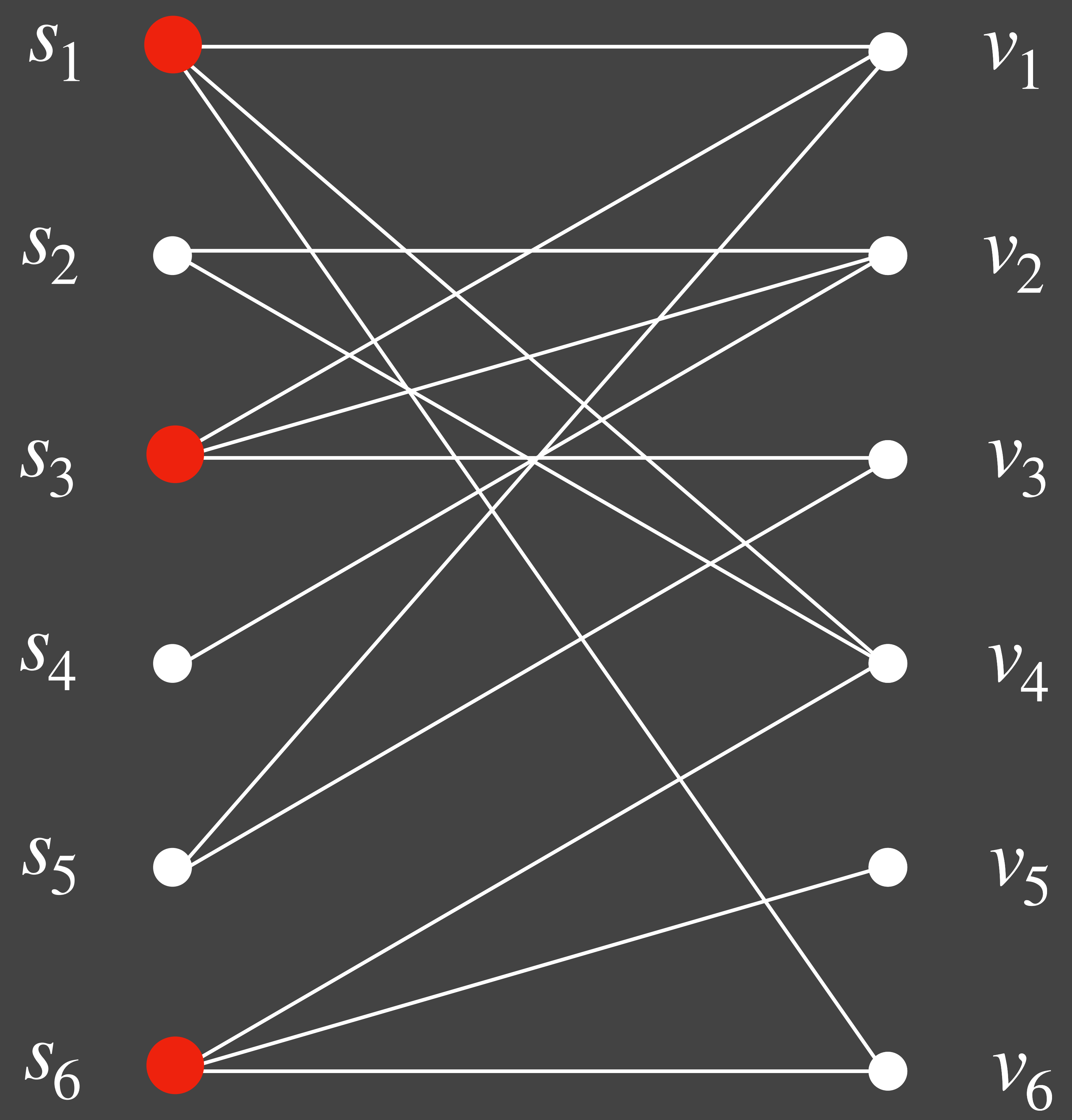


Running Example: Set Cover



Why should we care?

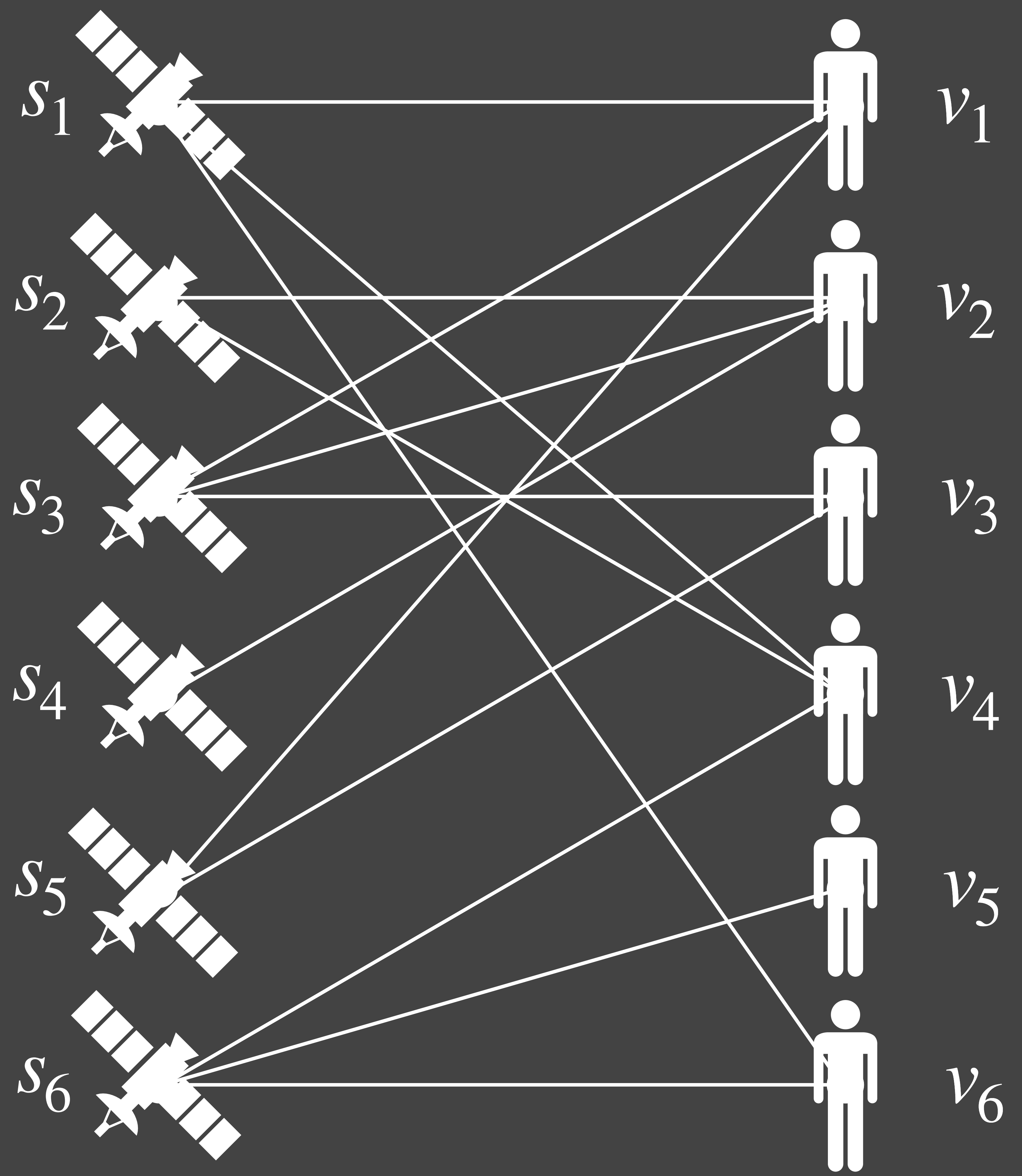
Running Example: Set Cover



Why should we care?

1. Natural applications to resource allocation.

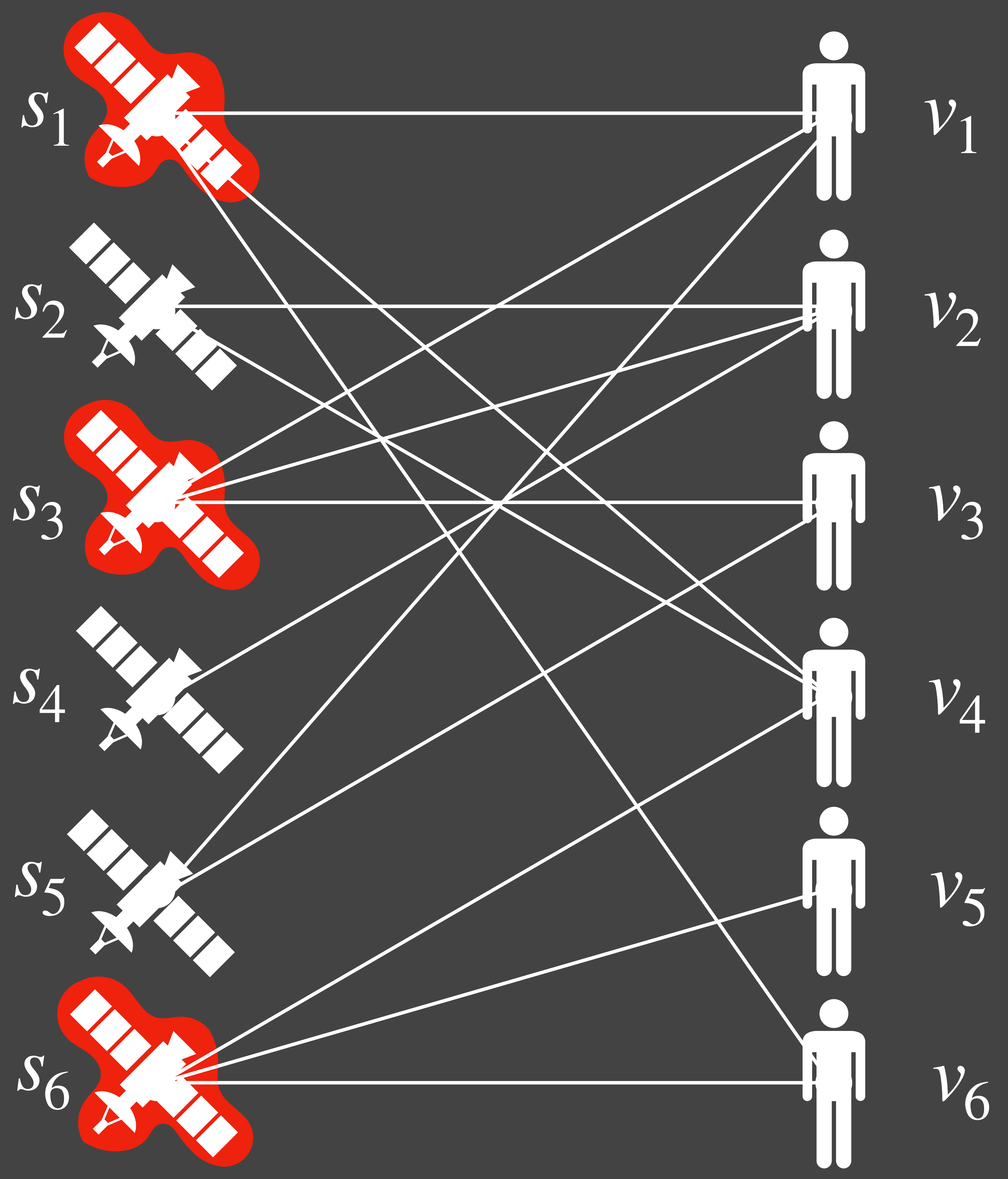
Running Example: Set Cover



Why should we care?

1. Natural applications to resource allocation.

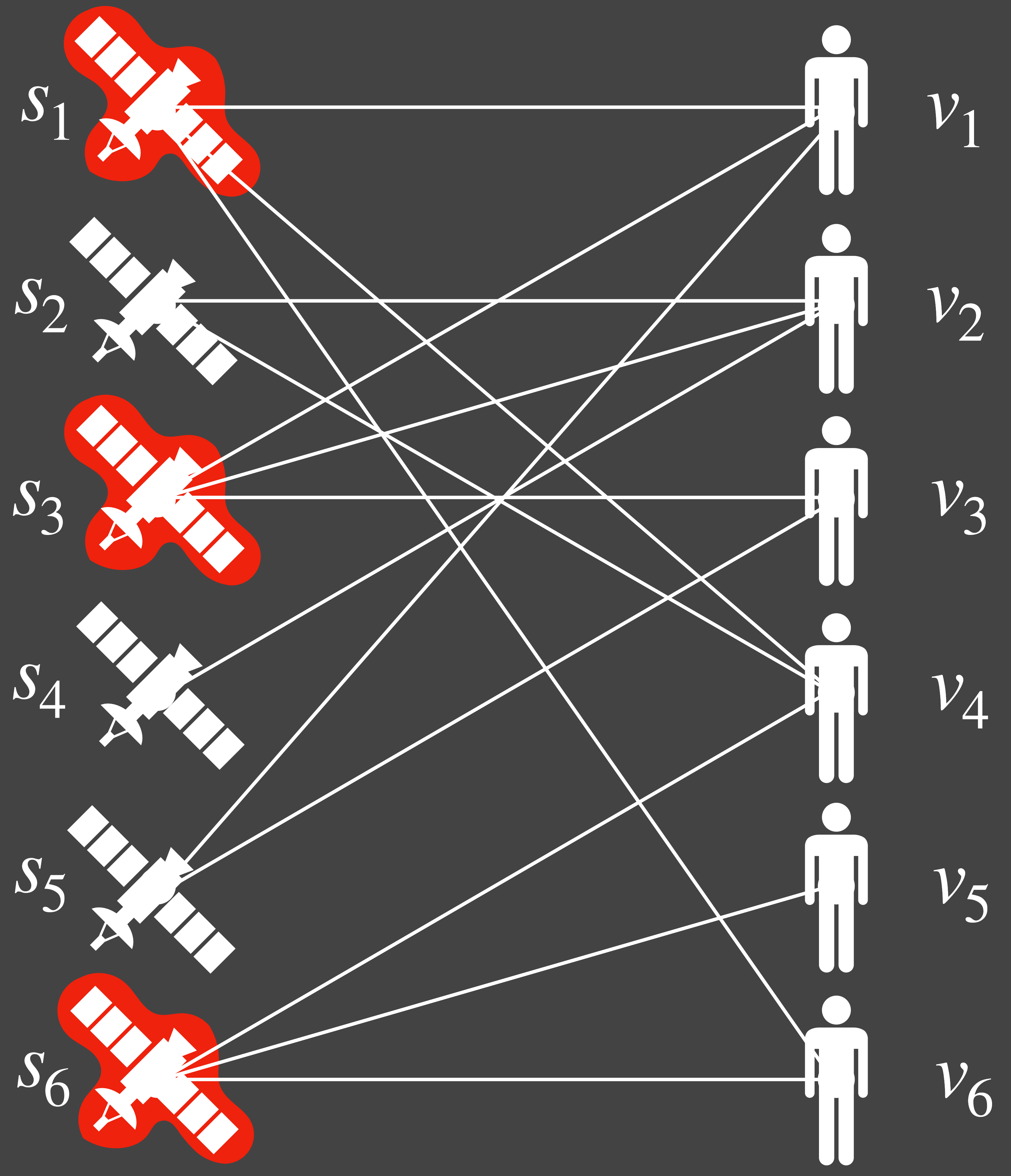
Running Example: Set Cover



Why should we care?

1. Natural applications to resource allocation.

Running Example: Set Cover

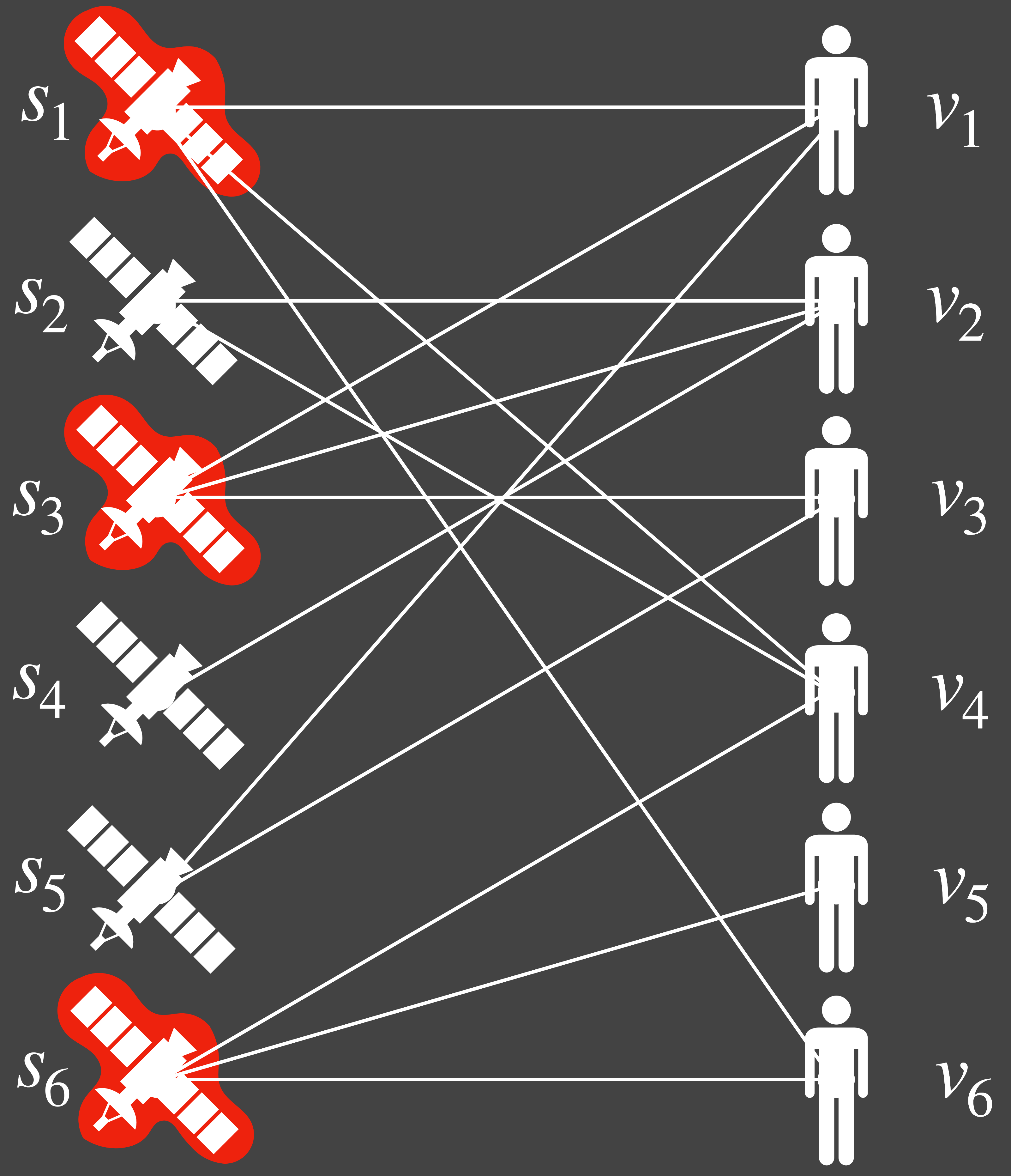


Why should we care?

1. Natural applications to resource allocation.

2. Sandbox for fundamental algorithmic ideas.

Running Example: Set Cover

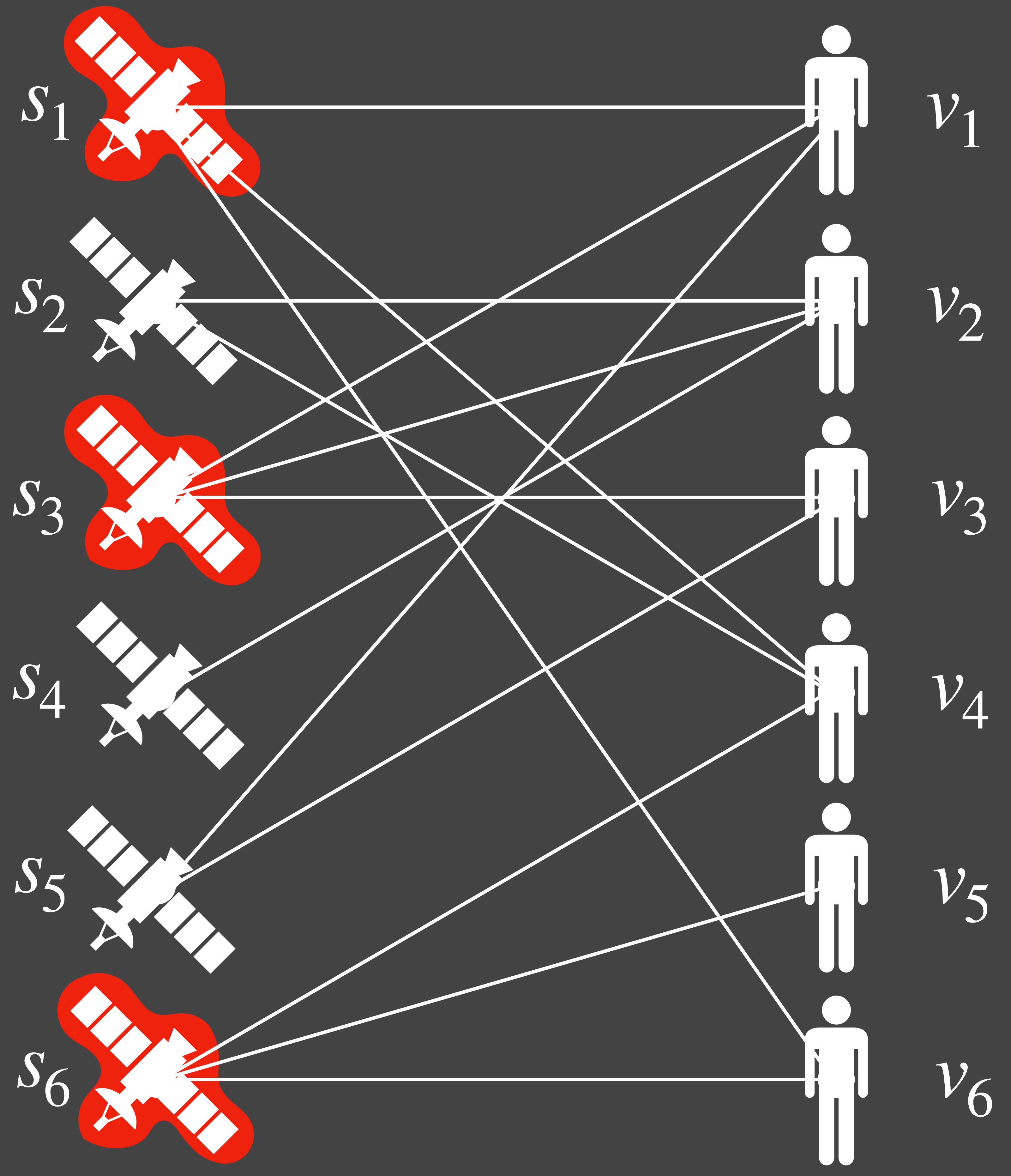


Why should we care?

1. Natural applications to resource allocation.
2. Sandbox for fundamental algorithmic ideas.

$$\min\ c^\intercal x$$
$$Ax \geq 1$$
$$x \in \mathbb{Z}^n_{\geq 0}$$

Running Example: Set Cover



Why should we care?

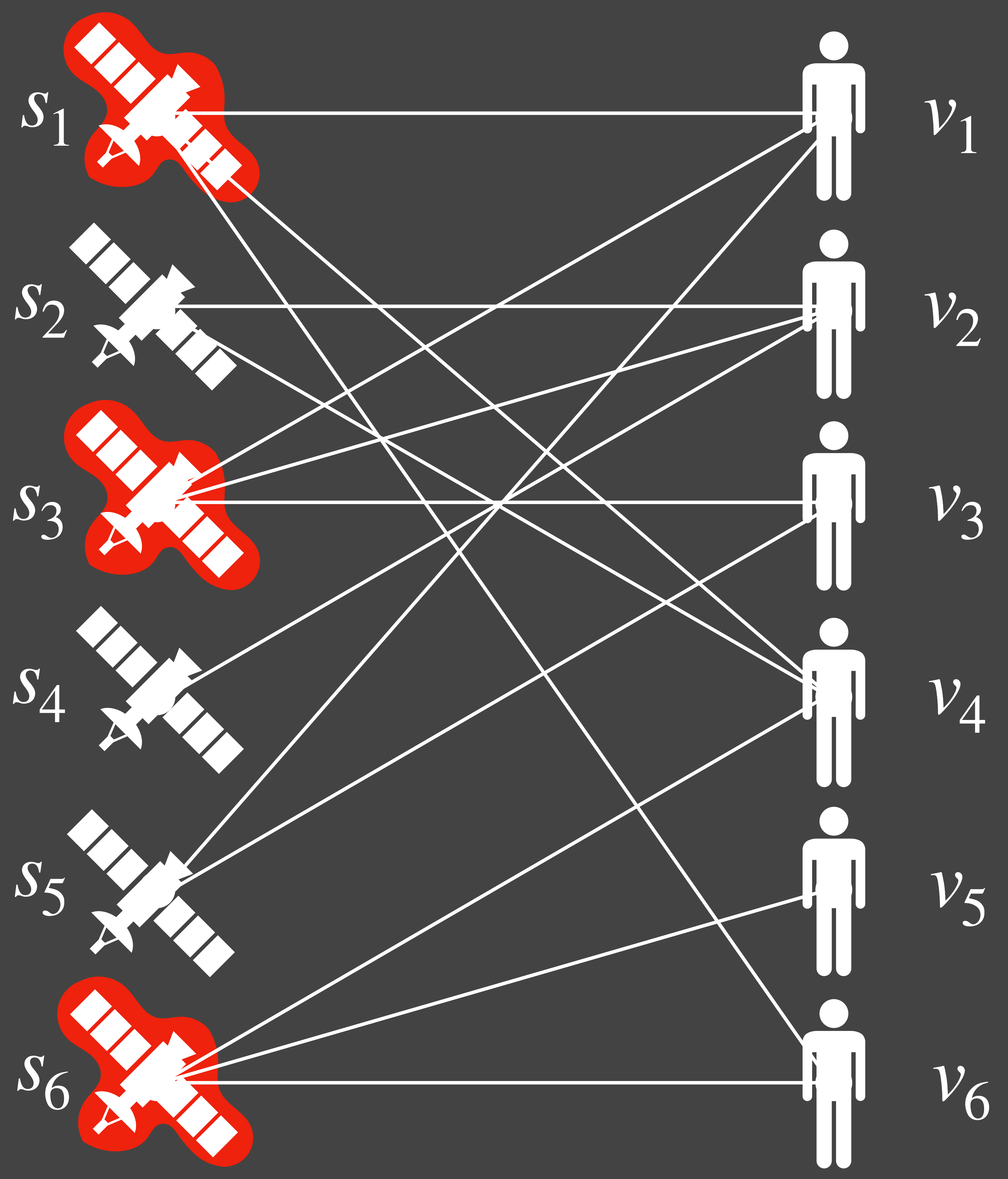
1. Natural applications to resource allocation.

2. Sandbox for fundamental algorithmic ideas.

$$\begin{aligned} &\min\ c^\intercal x \\ &Ax \geq 1 \\ &x \in \mathbb{Z}^n_{\geq 0} \end{aligned}$$

Special case of
Integer Programming
where A is 0/1.

Running Example: Set Cover



Why should we care?

1. Natural applications to resource allocation.

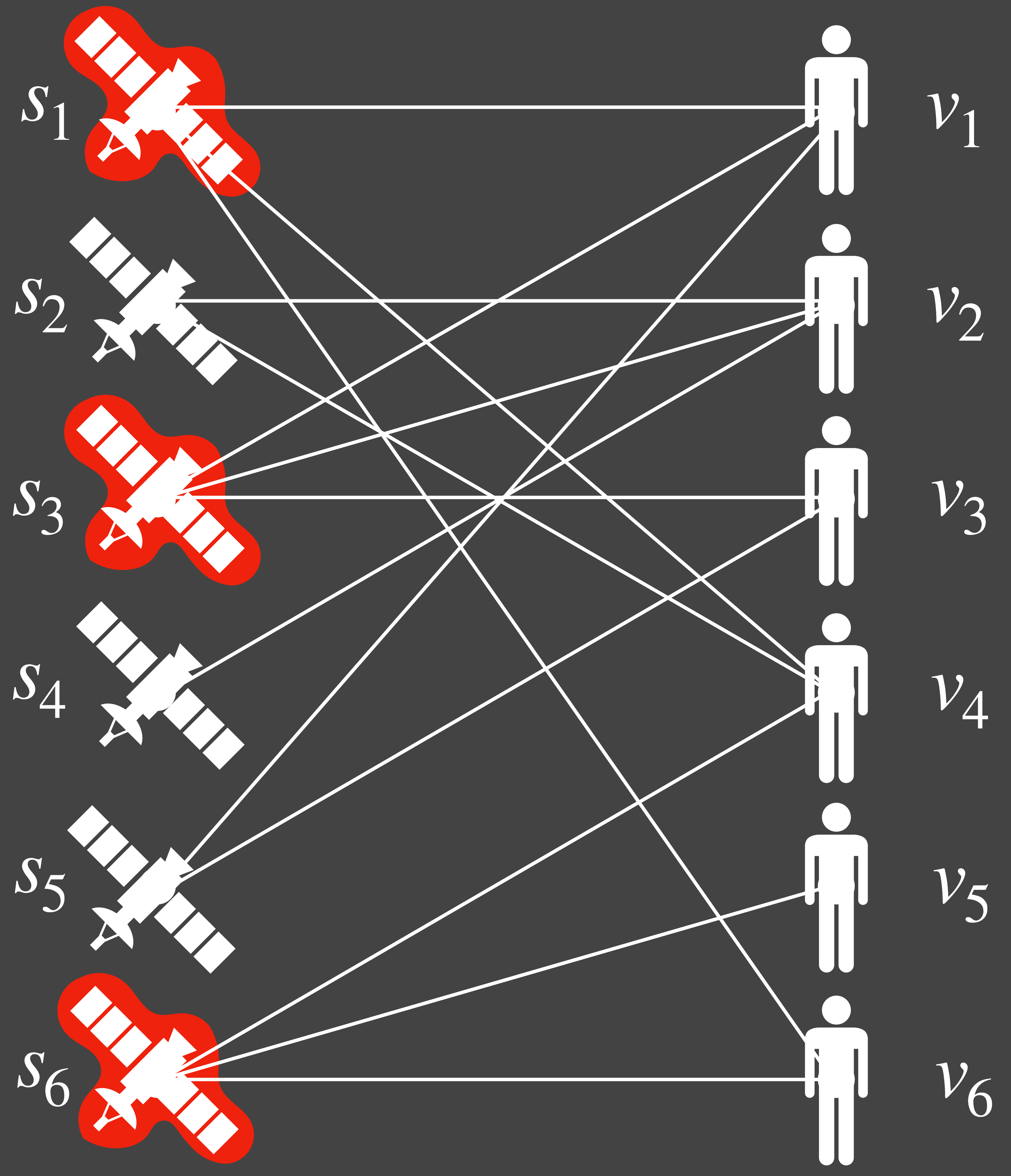
2. Sandbox for fundamental algorithmic ideas.

$$\min \; c^\intercal x$$
$$Ax \geq 1$$
$$x \in \mathbb{Z}^n_{\geq 0}$$

Special case of Integer Programming where A is 0/1.

Version 0 of EVERY discrete optimization problem!

Running Example: Set Cover



Why should we care?

1. Natural applications to resource allocation.

2. Sandbox for fundamental algorithmic ideas.

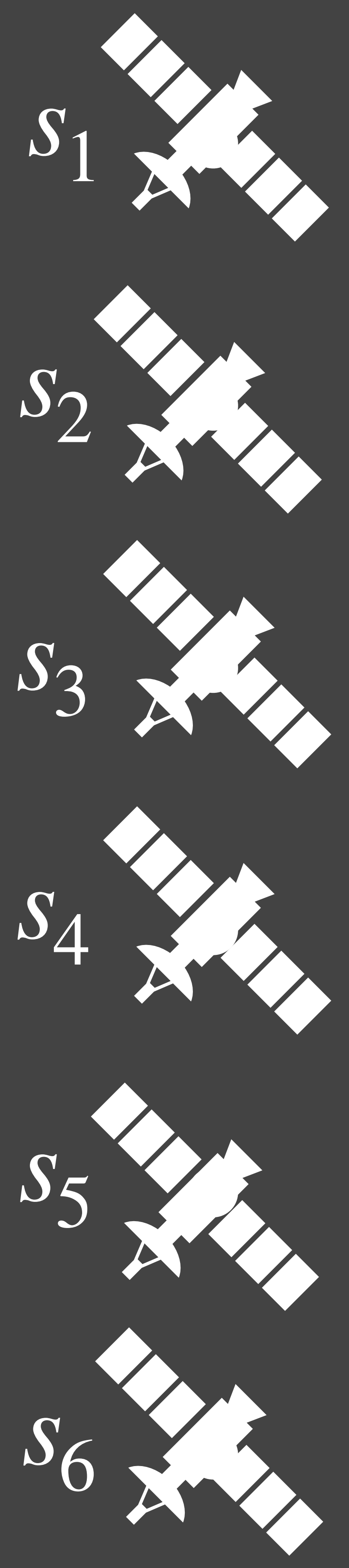
$$\min \; c^\intercal x$$
$$Ax \geq 1$$
$$x \in \mathbb{Z}^n_{\geq 0}$$

Special case of Integer Programming where A is 0/1.

Version 0 of EVERY discrete optimization problem!

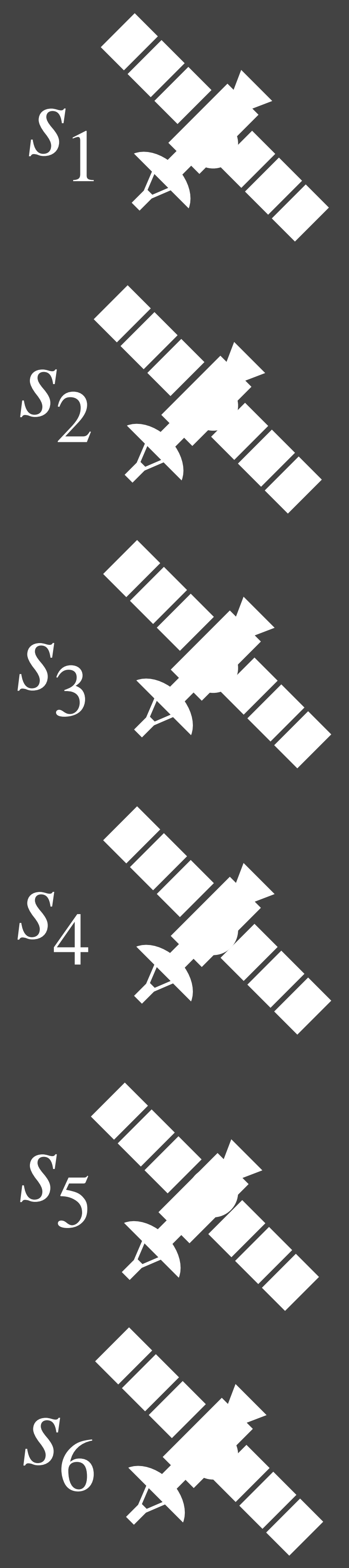
3. Fast algos get good approximation: $O(\log n)$ [Johnson 74], [Lovasz 75], [Chvatal 79]

Running Example: Set Cover



What if we don’t know user demand a-priori?

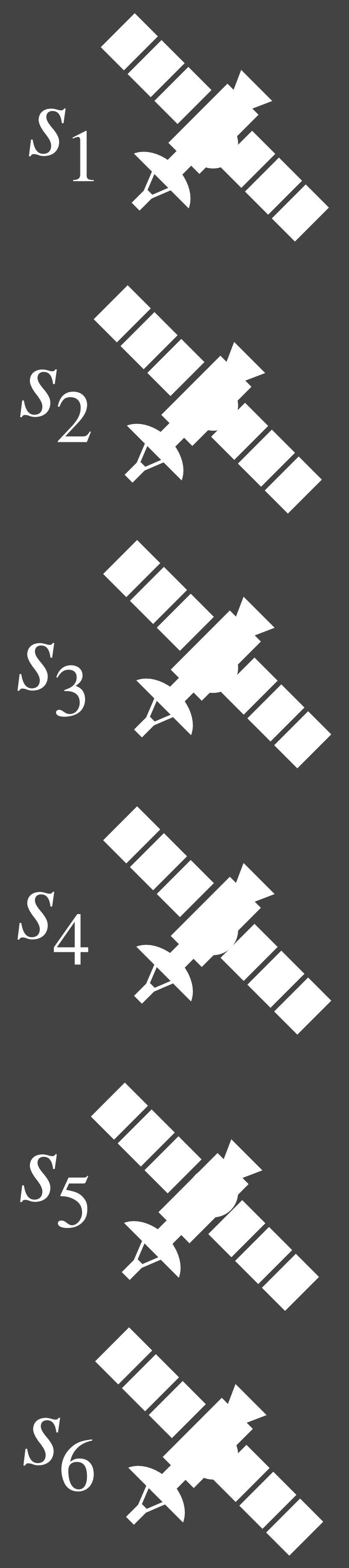
Running Example: Set Cover



What if we don’t know user demand a-priori?

Requests arrive over time, need to satisfy immediately.

Running Example: Set Cover

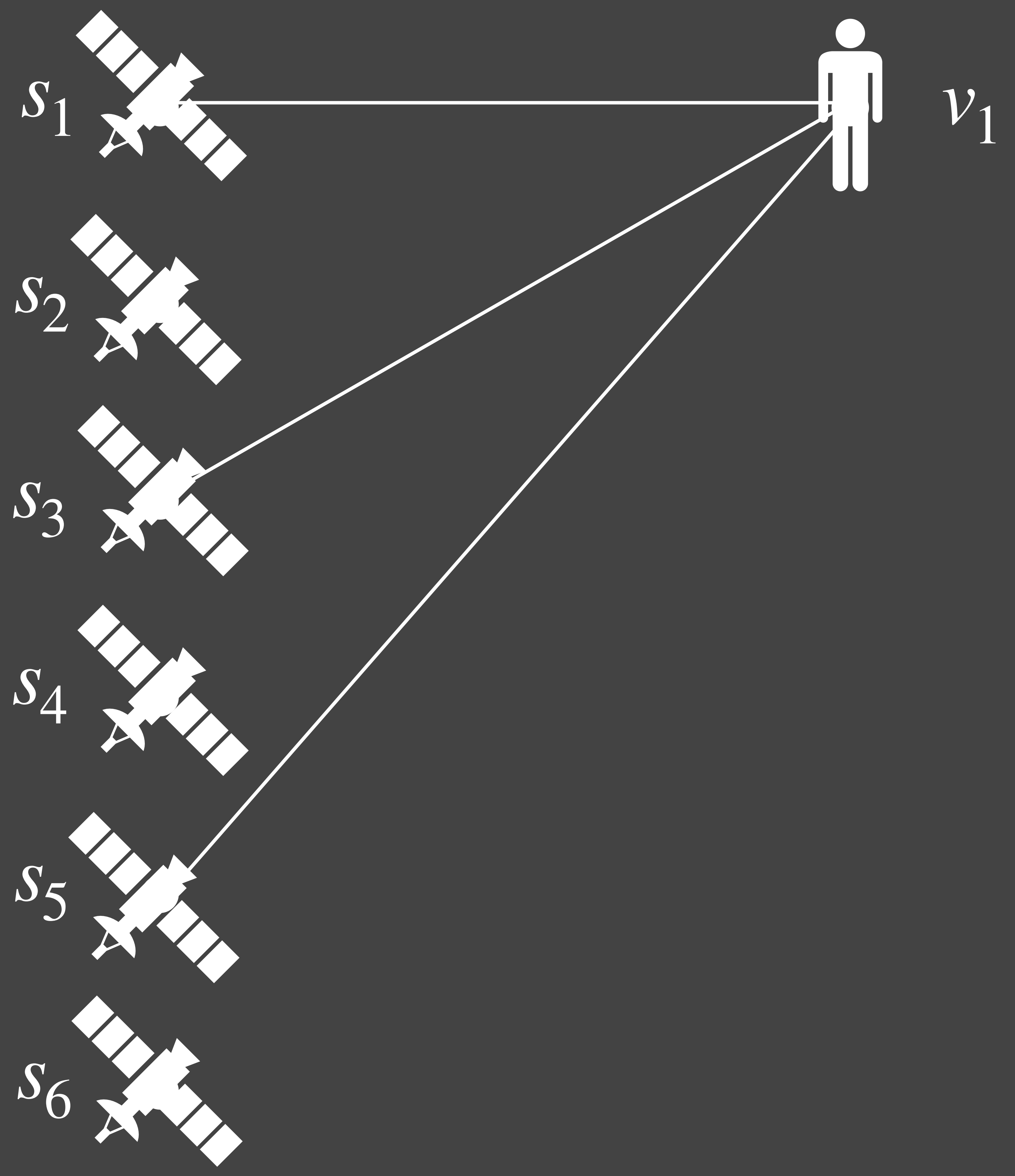


What if we don't know user demand a-priori?

Requests arrive over time, need to satisfy immediately.

Expensive to open satellites!
Model decisions as irrevocable.

Running Example: Set Cover

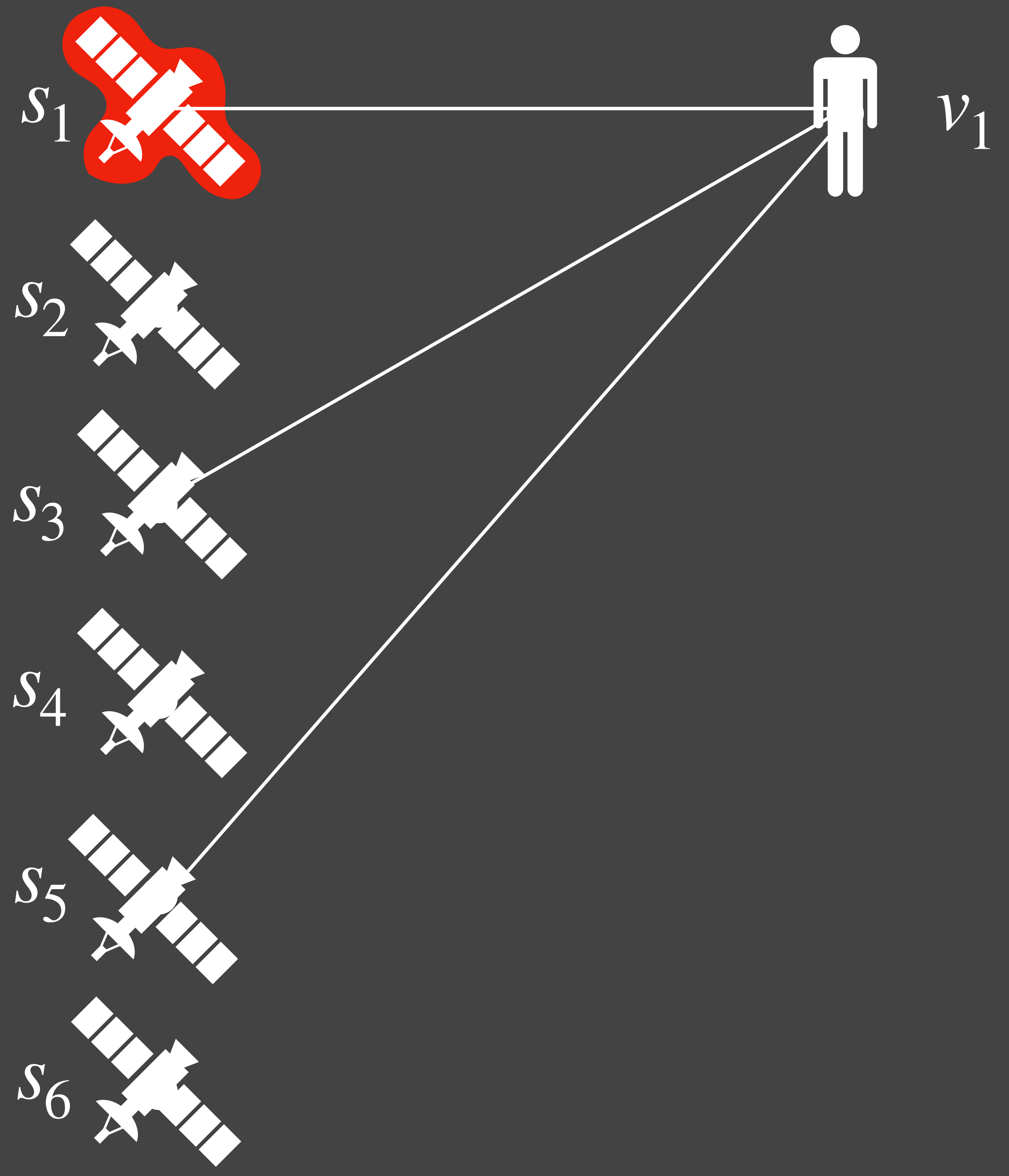


What if we don't know user demand a-priori?

Requests arrive over time, need to satisfy immediately.

Expensive to open satellites!
Model decisions as irrevocable.

Running Example: Set Cover

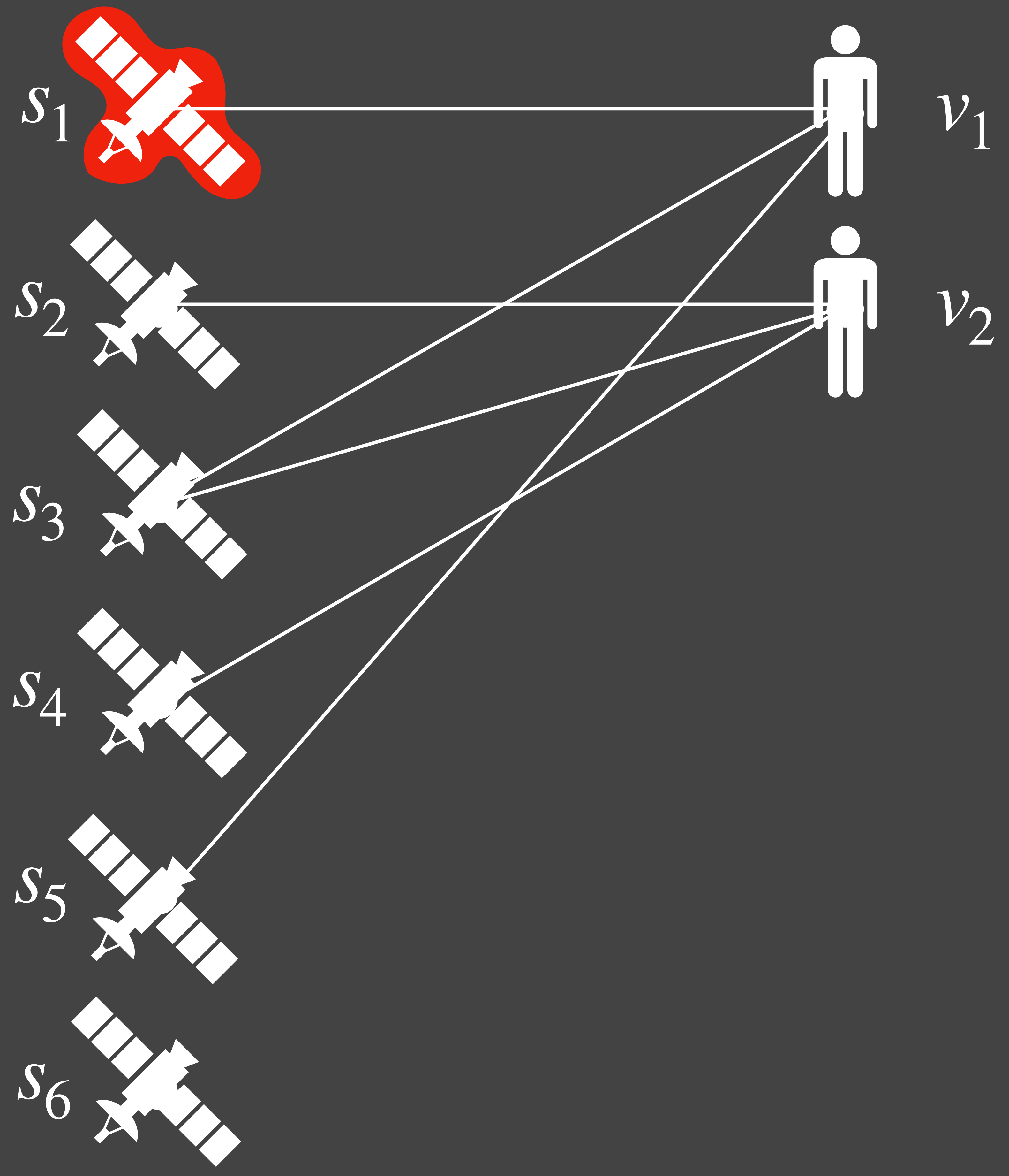


What if we don't know user demand a-priori?

Requests arrive over time, need to satisfy immediately.

Expensive to open satellites!
Model decisions as irrevocable.

Running Example: Set Cover

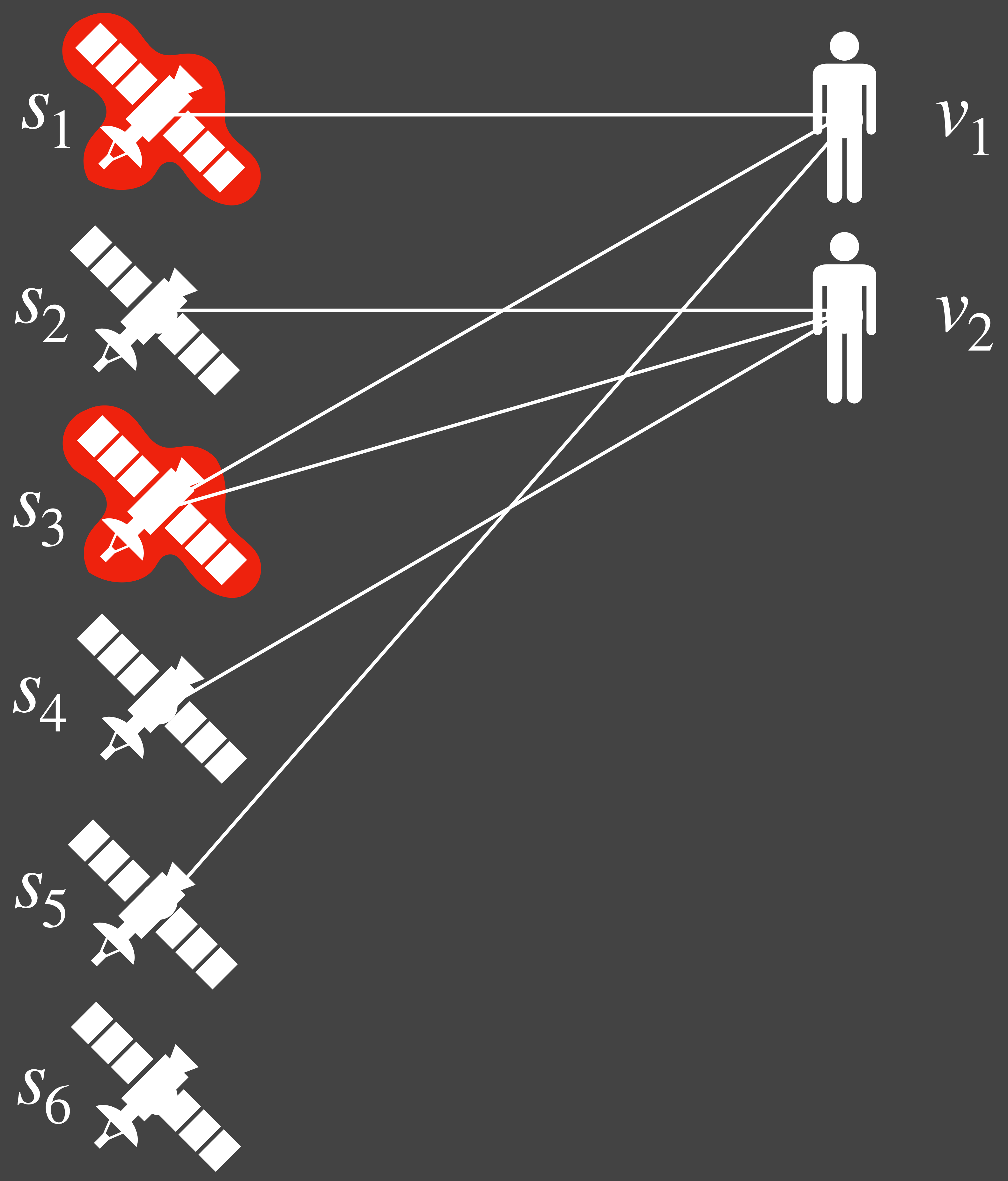


What if we don’t know user demand a-priori?

Requests arrive over time, need to satisfy immediately.

Expensive to open satellites!
Model decisions as irrevocable.

Running Example: Set Cover

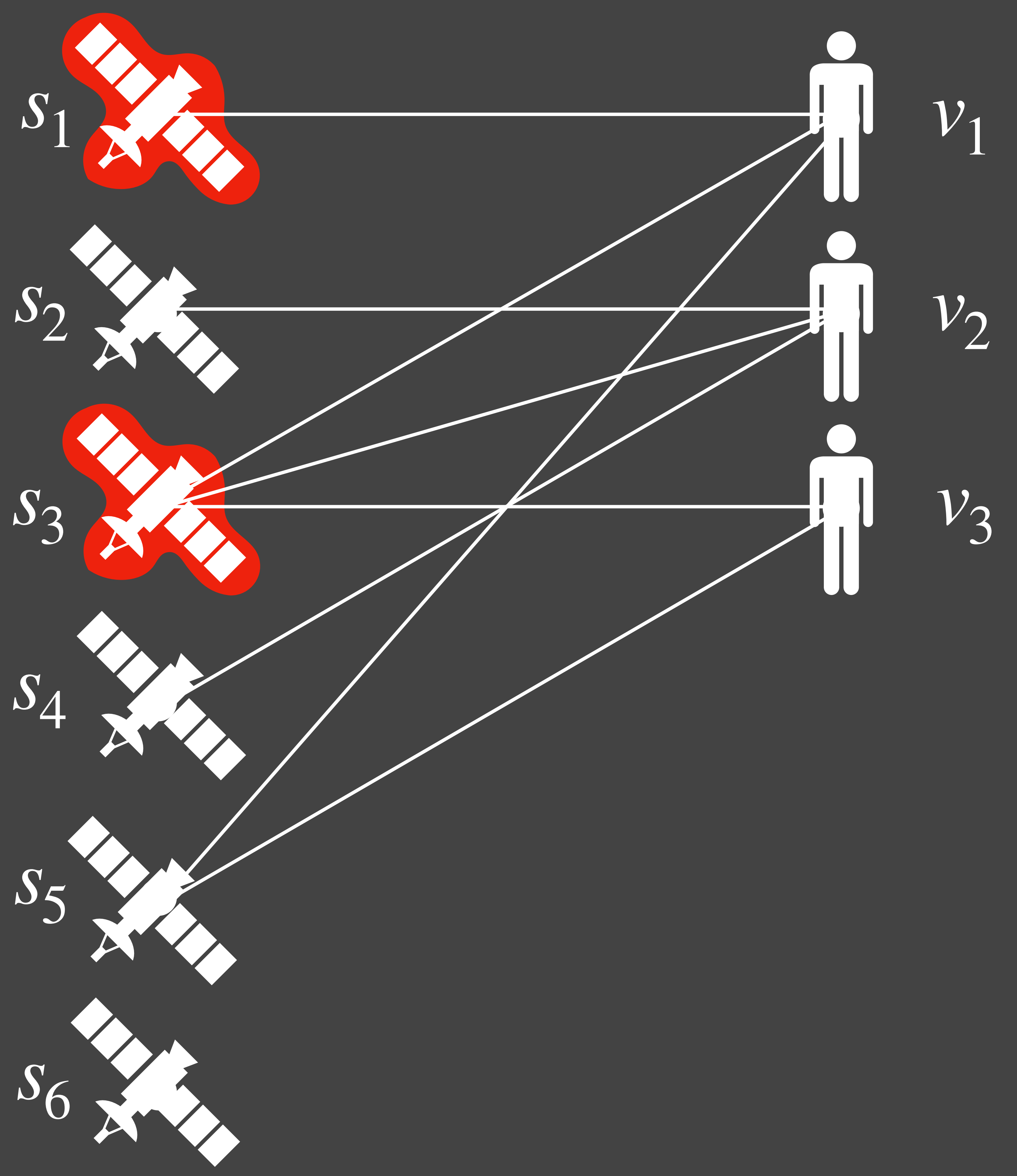


What if we don’t know user user demand a-priori?

Requests arrive over time, need to satisfy immediately.

Expensive to open satellites!
Model decisions as irrevocable.

Running Example: Set Cover

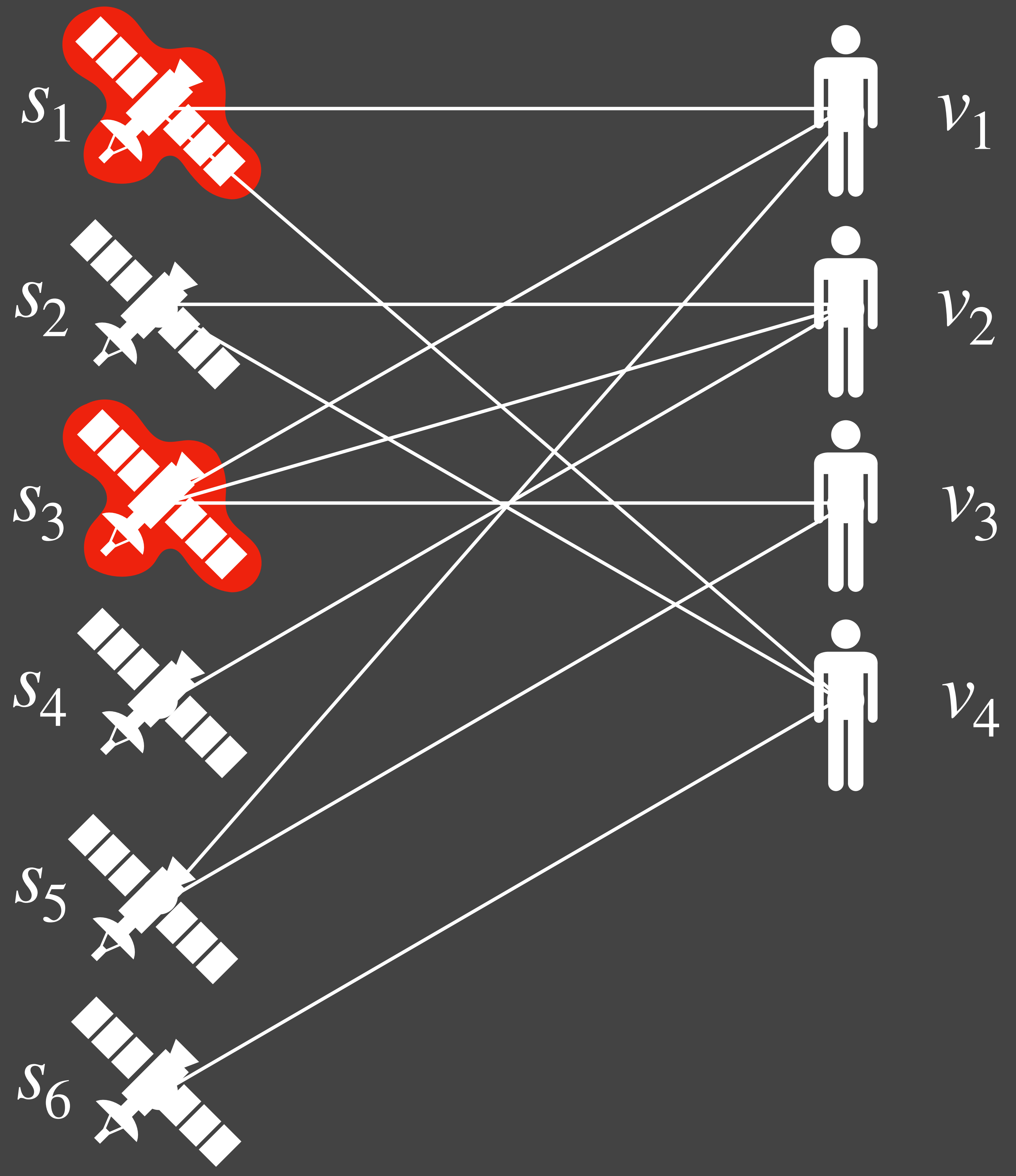


What if we don’t know user demand a-priori?

Requests arrive over time, need to satisfy immediately.

Expensive to open satellites!
Model decisions as irrevocable.

Running Example: Set Cover

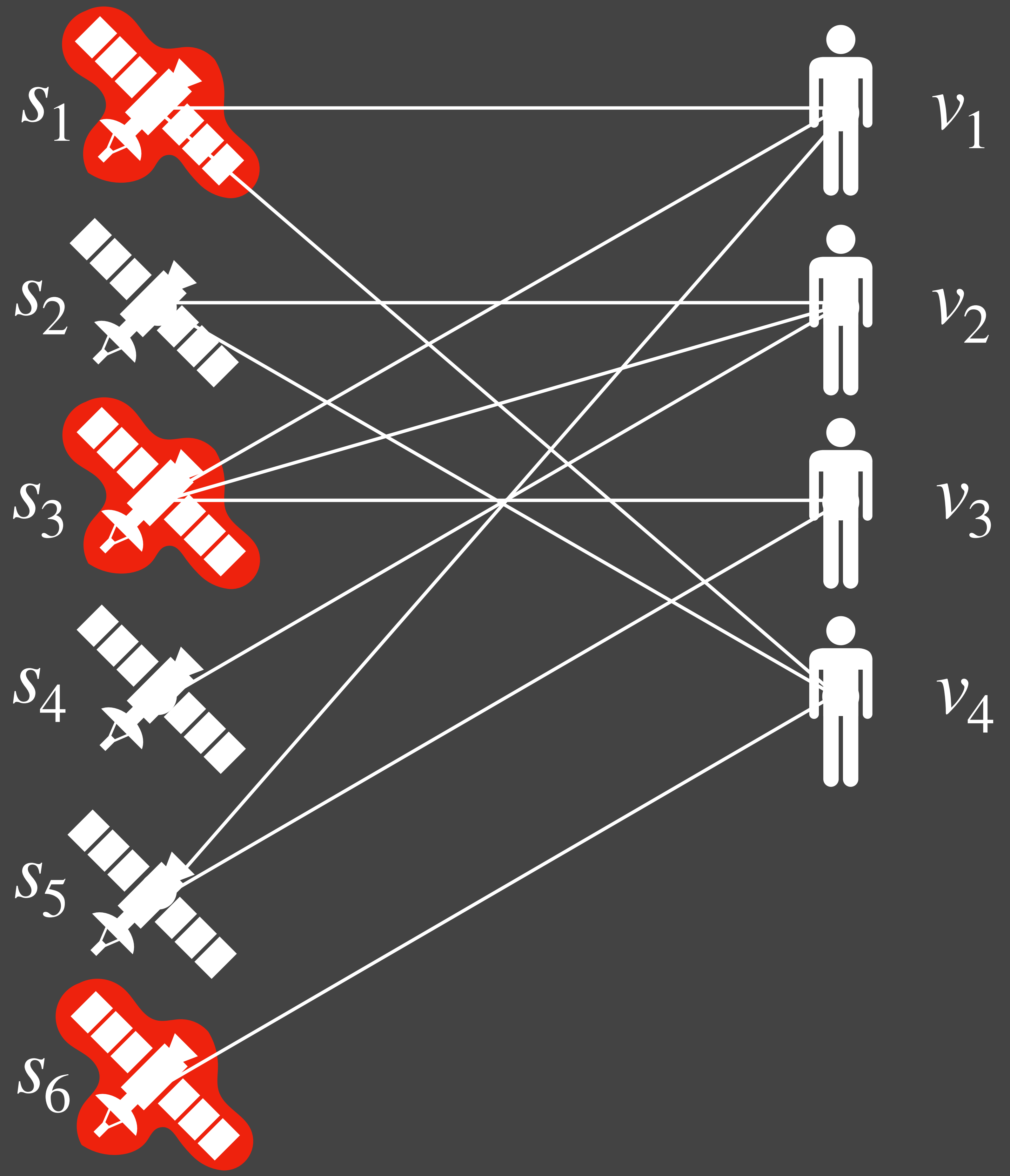


What if we don’t know user demand a-priori?

Requests arrive over time, need to satisfy immediately.

Expensive to open satellites!
Model decisions as irrevocable.

Running Example: Set Cover

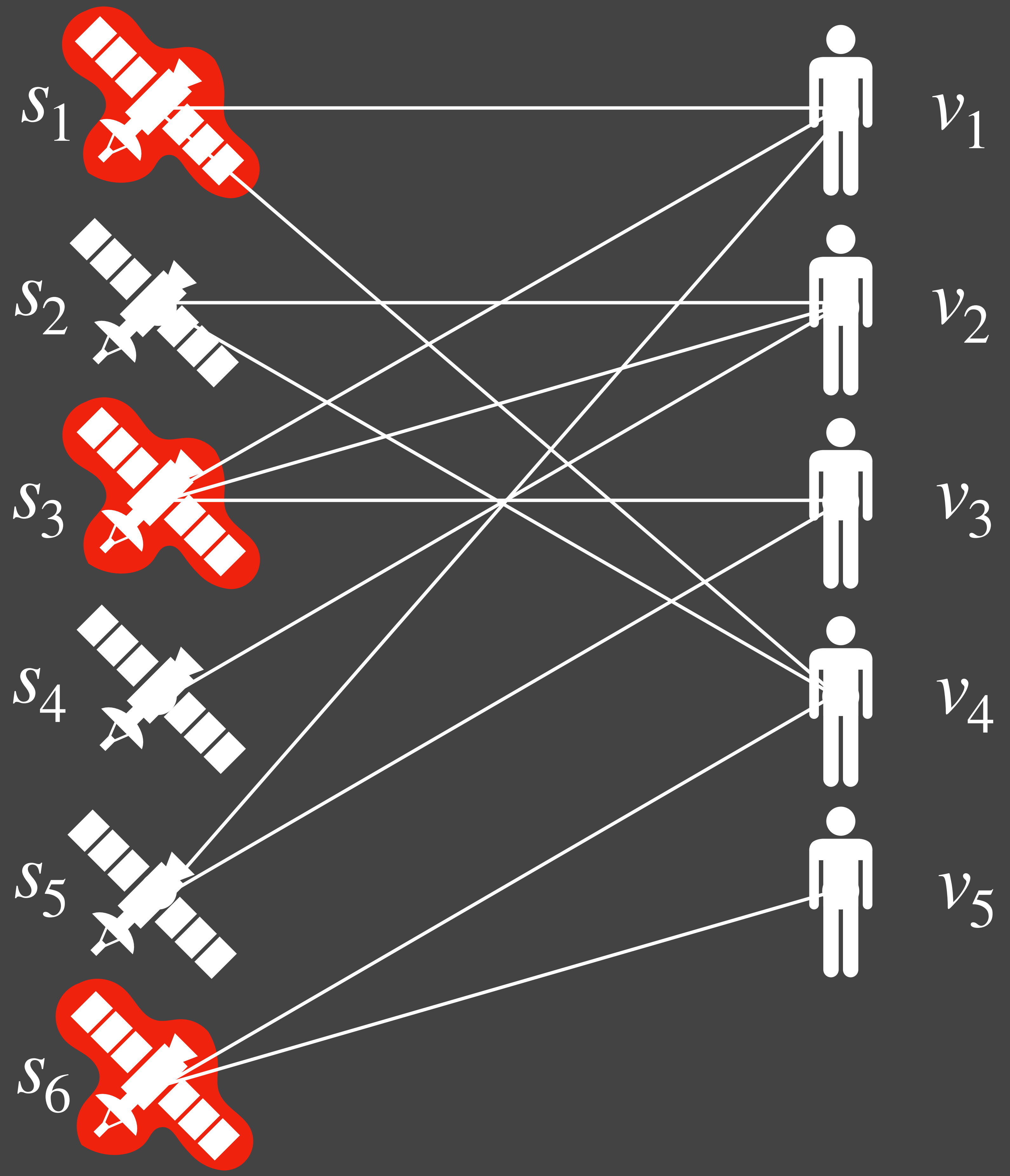


What if we don't know user demand a-priori?

Requests arrive over time, need to satisfy immediately.

Expensive to open satellites!
Model decisions as irrevocable.

Running Example: Set Cover

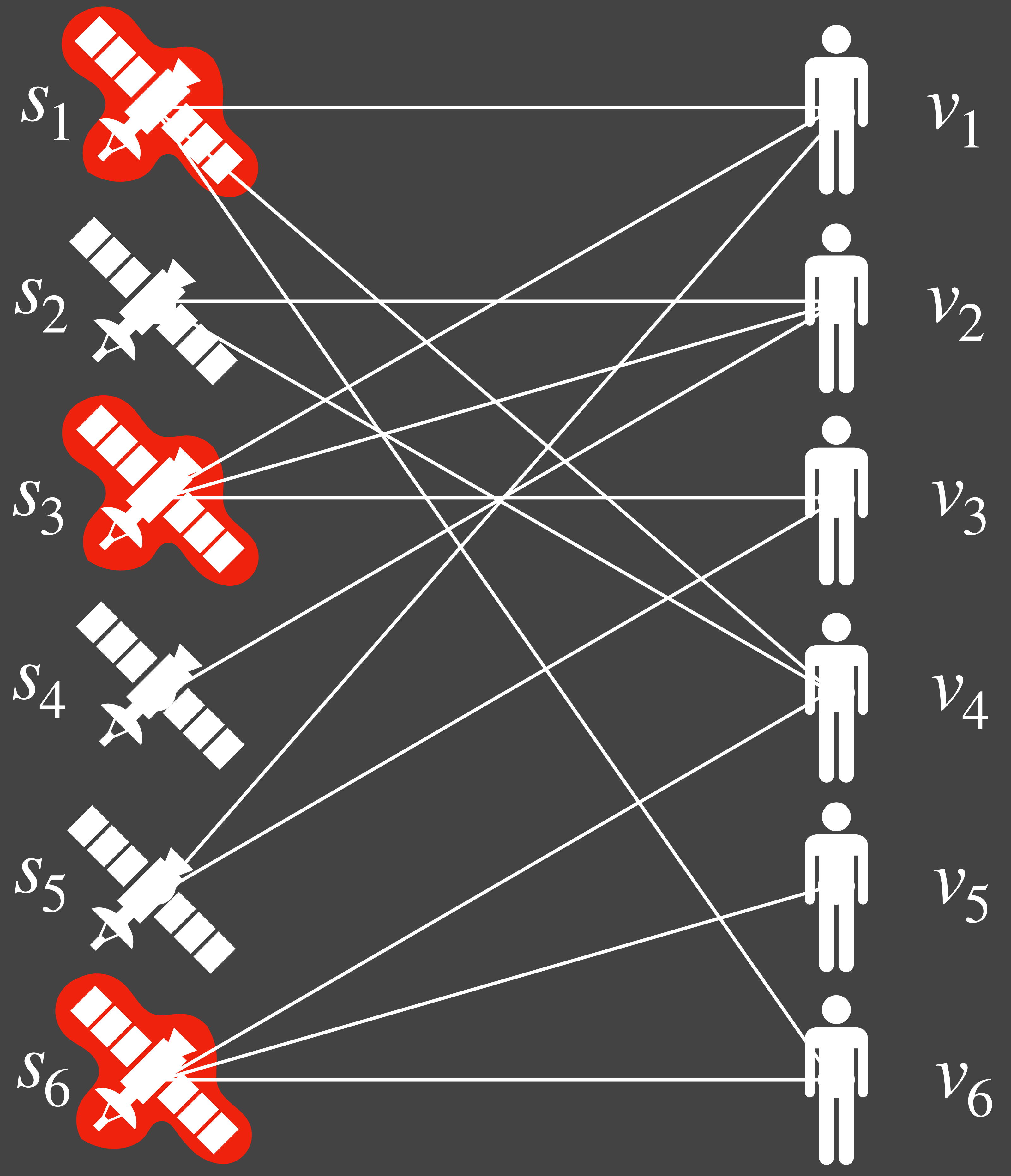


What if we don't know user demand a-priori?

Requests arrive over time, need to satisfy immediately.

Expensive to open satellites!
Model decisions as irrevocable.

Running Example: Set Cover

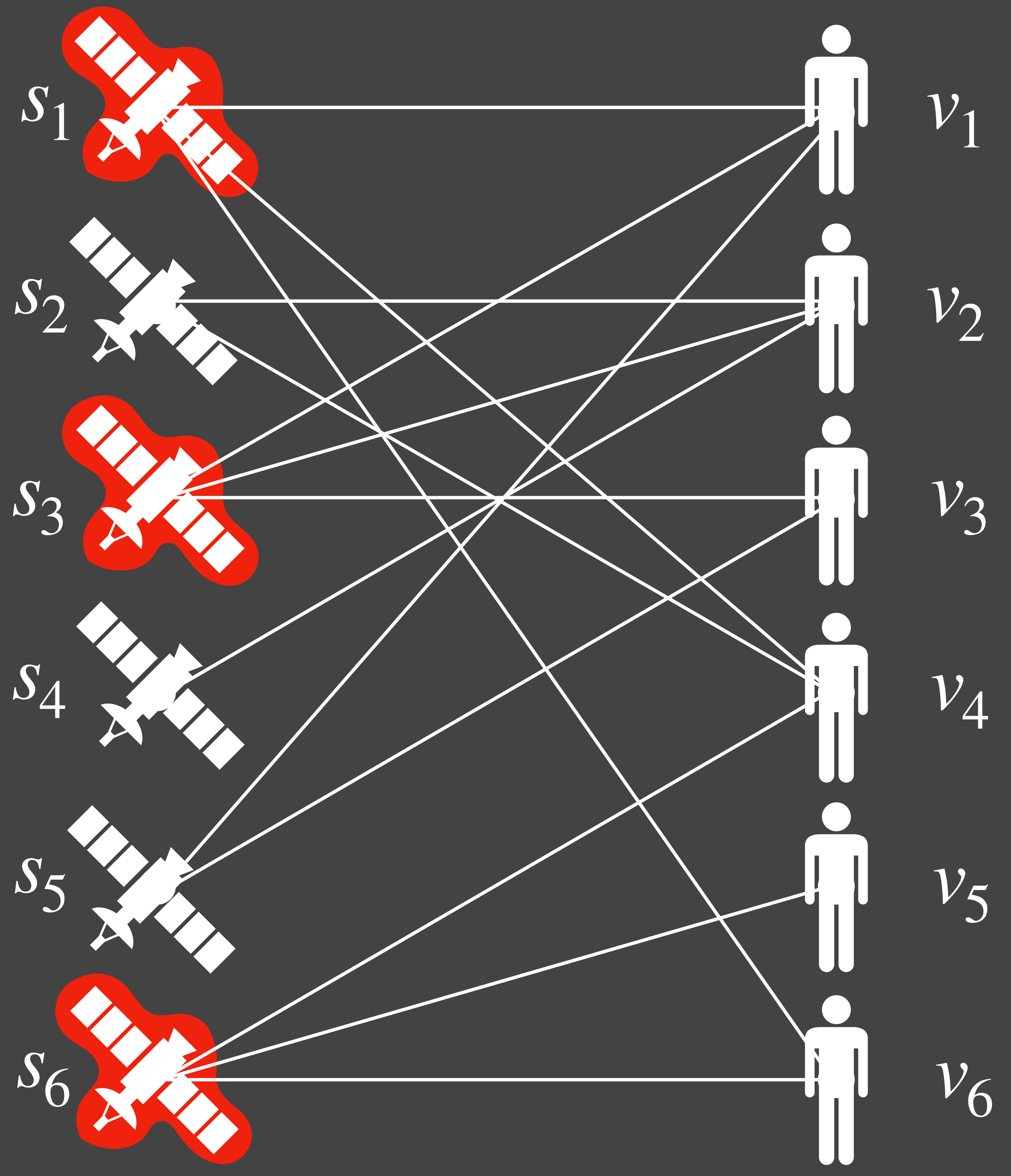


What if we don’t know user demand a-priori?

Requests arrive over time, need to satisfy immediately.

Expensive to open satellites!
Model decisions as irrevocable.

Running Example: Set Cover



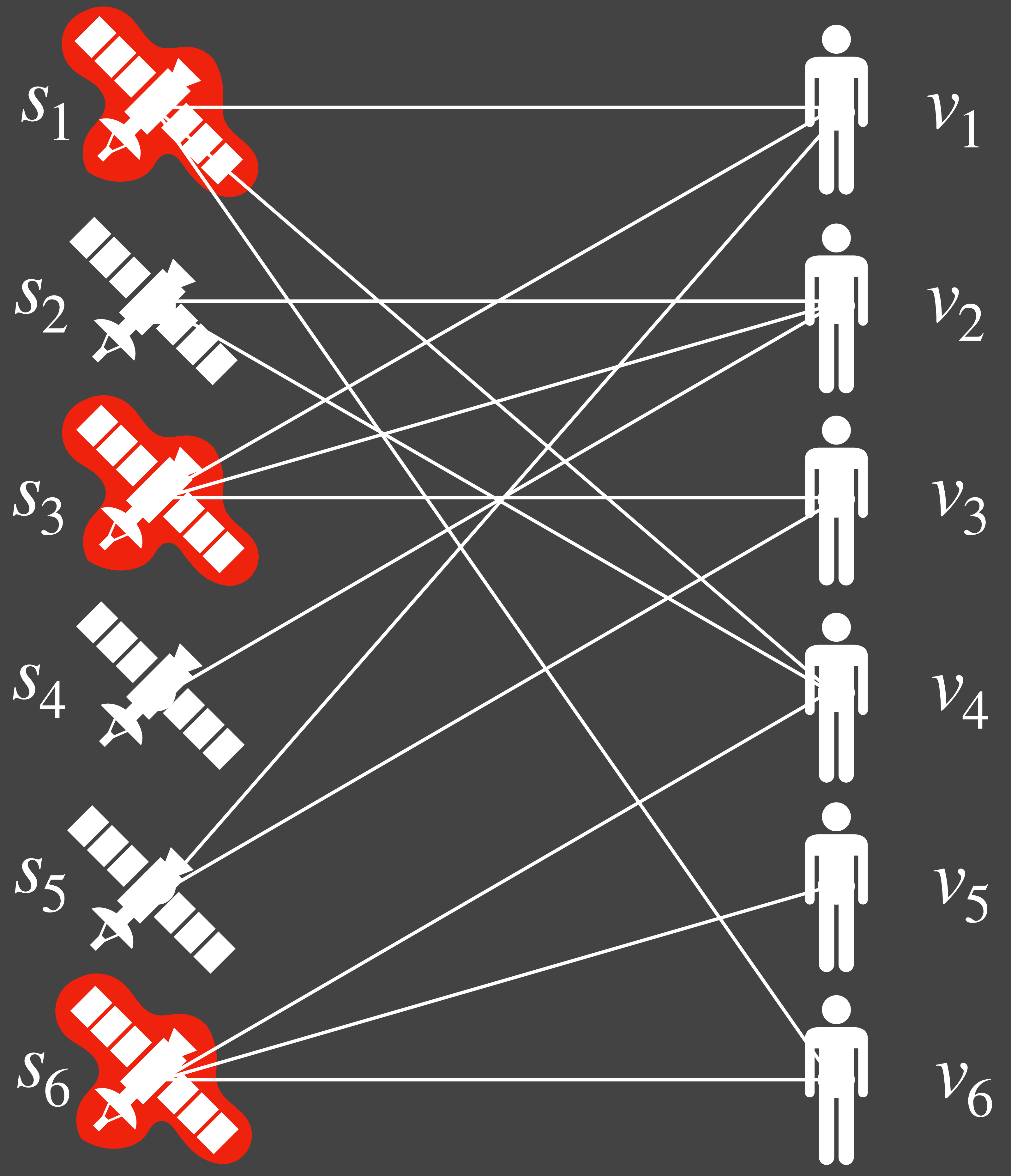
What if we don't know user demand a-priori?

Requests arrive over time, need to satisfy immediately.

Expensive to open satellites!
Model decisions as irrevocable.

Q: Can we get good approximation, efficiently, despite not knowing the future?

Running Example: Set Cover



What if we don't know user demand a-priori?

Requests arrive over time, need to satisfy immediately.

Expensive to open satellites!
Model decisions as irrevocable.

Q: Can we get good approximation, efficiently, despite not knowing the future?

A: Yes! Approximation: $O(\log^2 n)$ [Alon Awerbuch Azar Buchbinder Naor 03] [Buchbinder Naor 09], this is optimal for polynomial time algorithms.

My Work

Online | Dynamic | Streaming

My Work

Online

Dynamic

Streaming

No take-backs

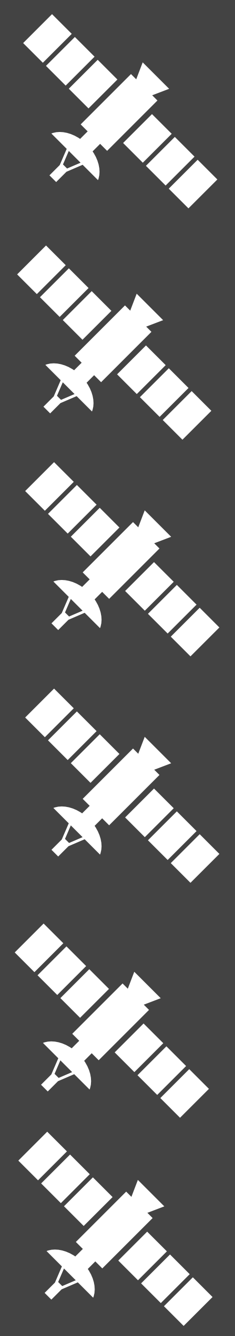
My Work

Online

Dynamic

Streaming

No take-backs



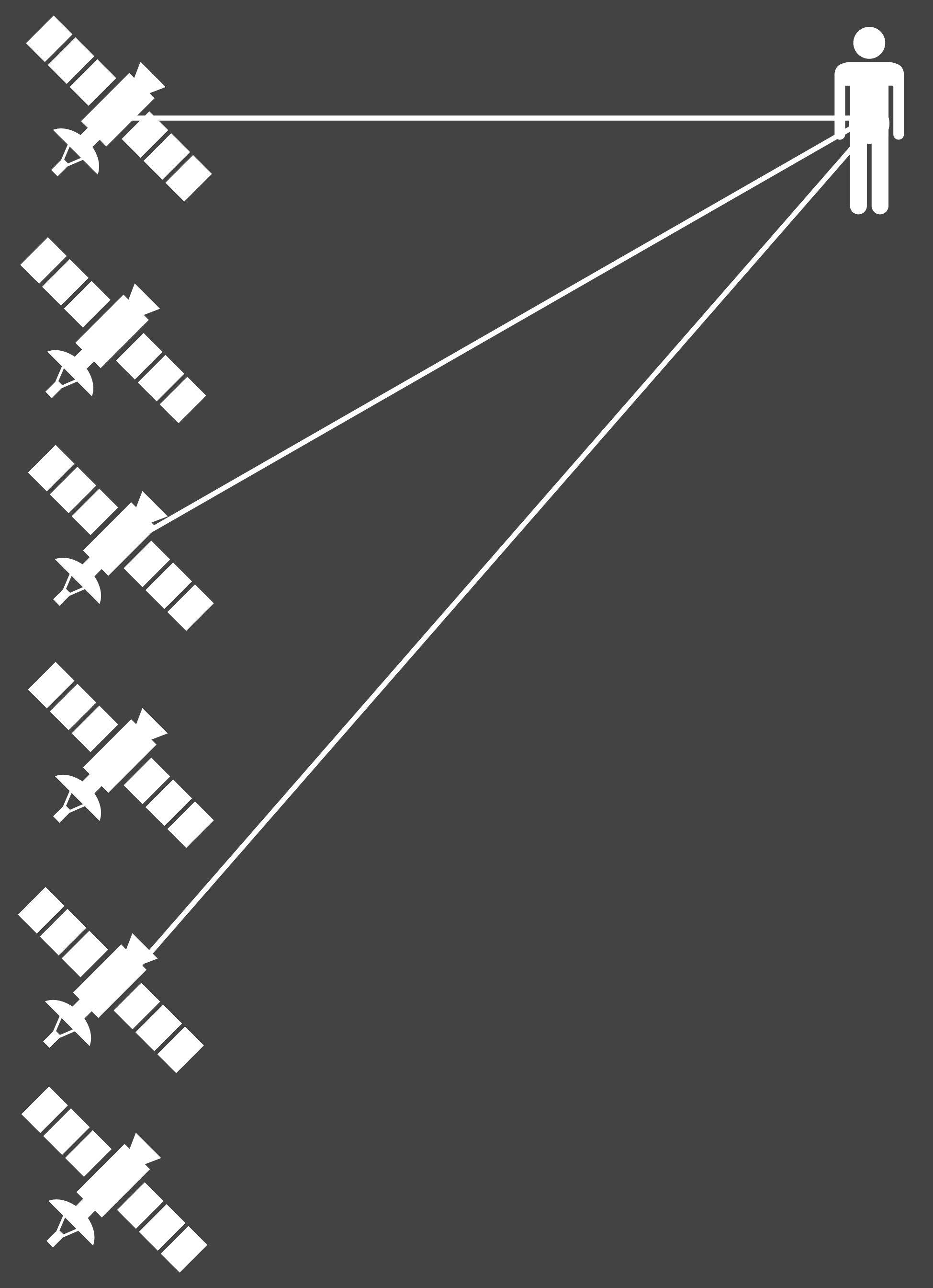
My Work

Online

Dynamic

Streaming

No take-backs



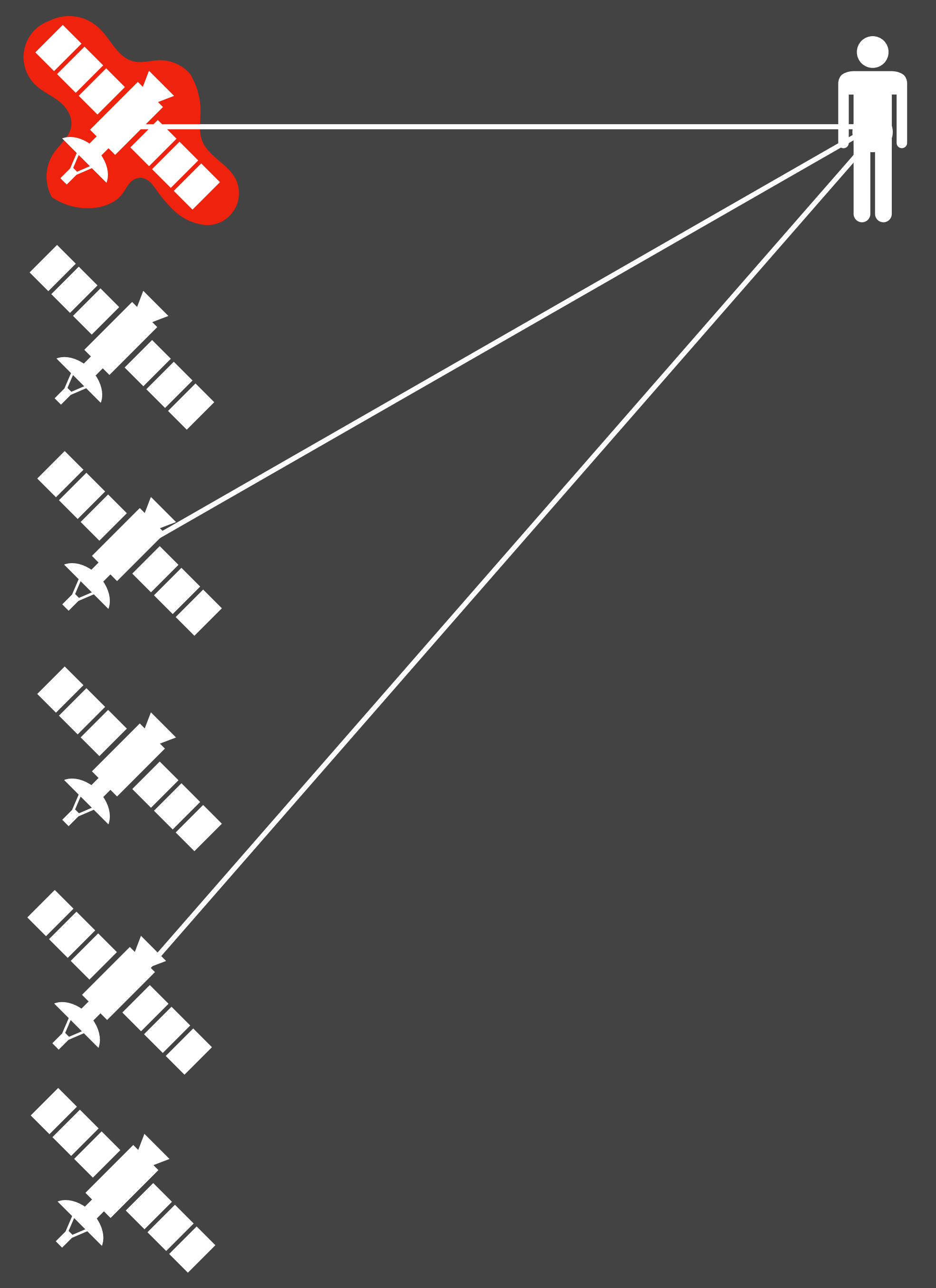
My Work

Online

Dynamic

Streaming

No take-backs



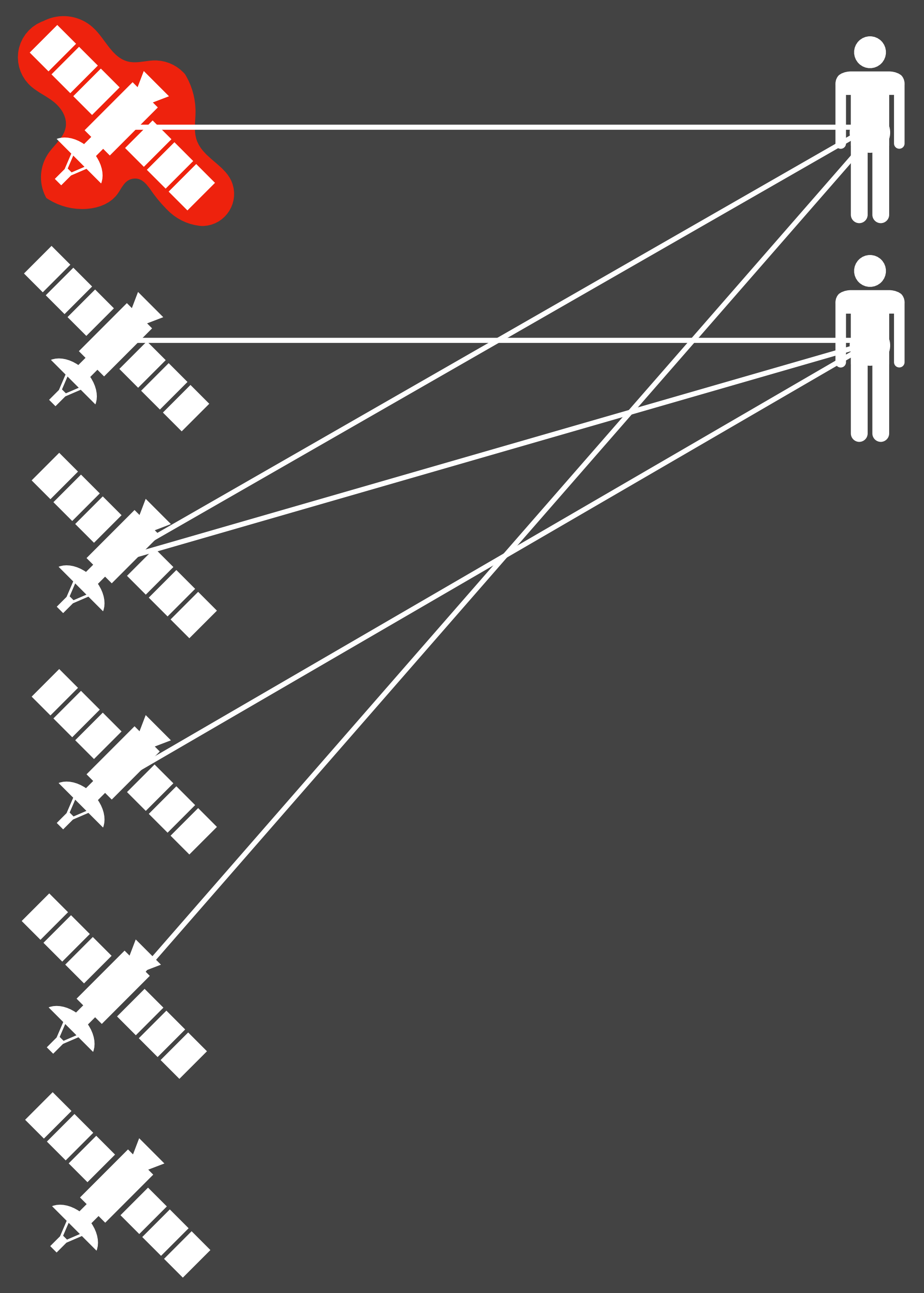
My Work

Online

Dynamic

Streaming

No take-backs



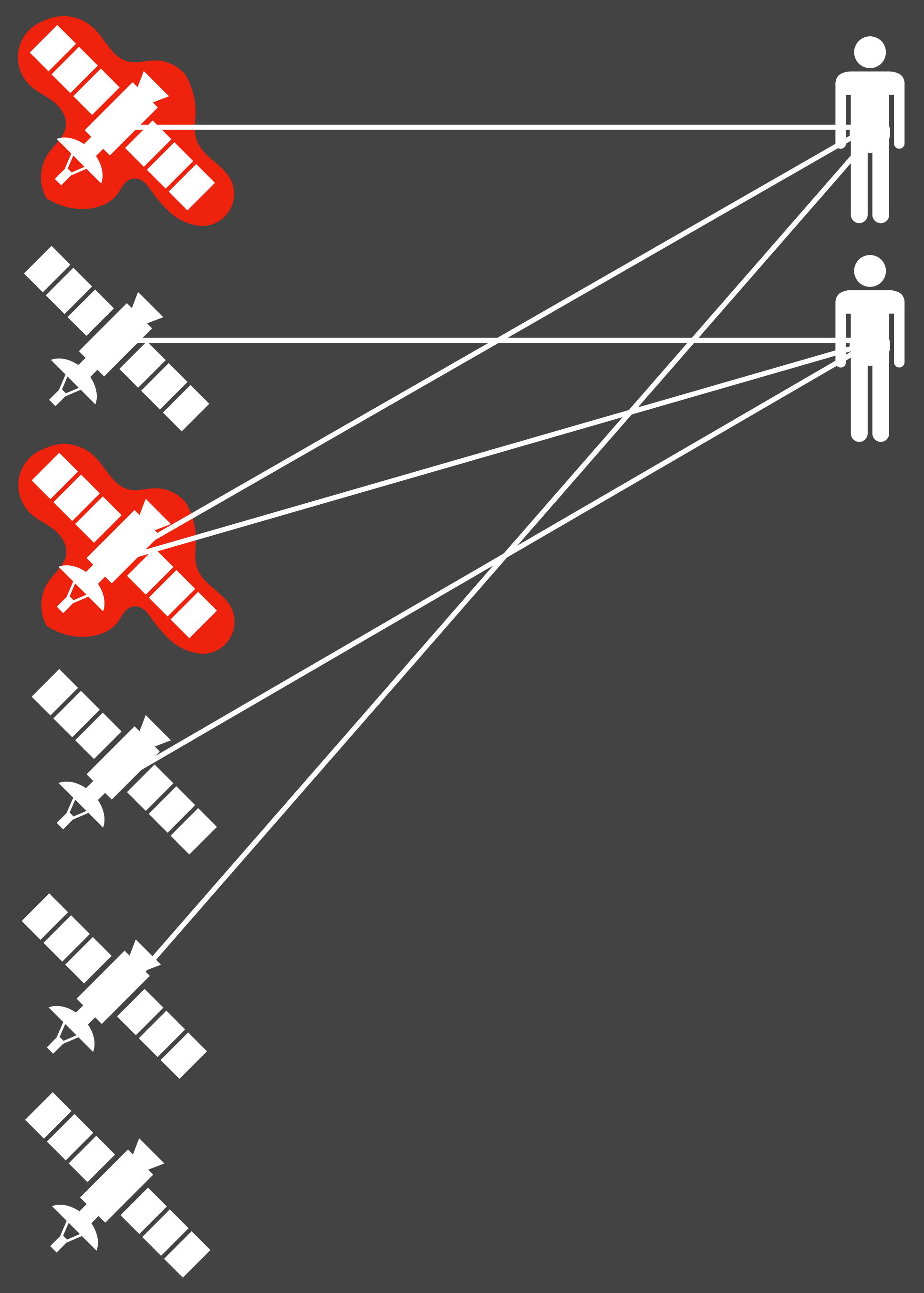
My Work

Online

Dynamic

Streaming

No take-backs



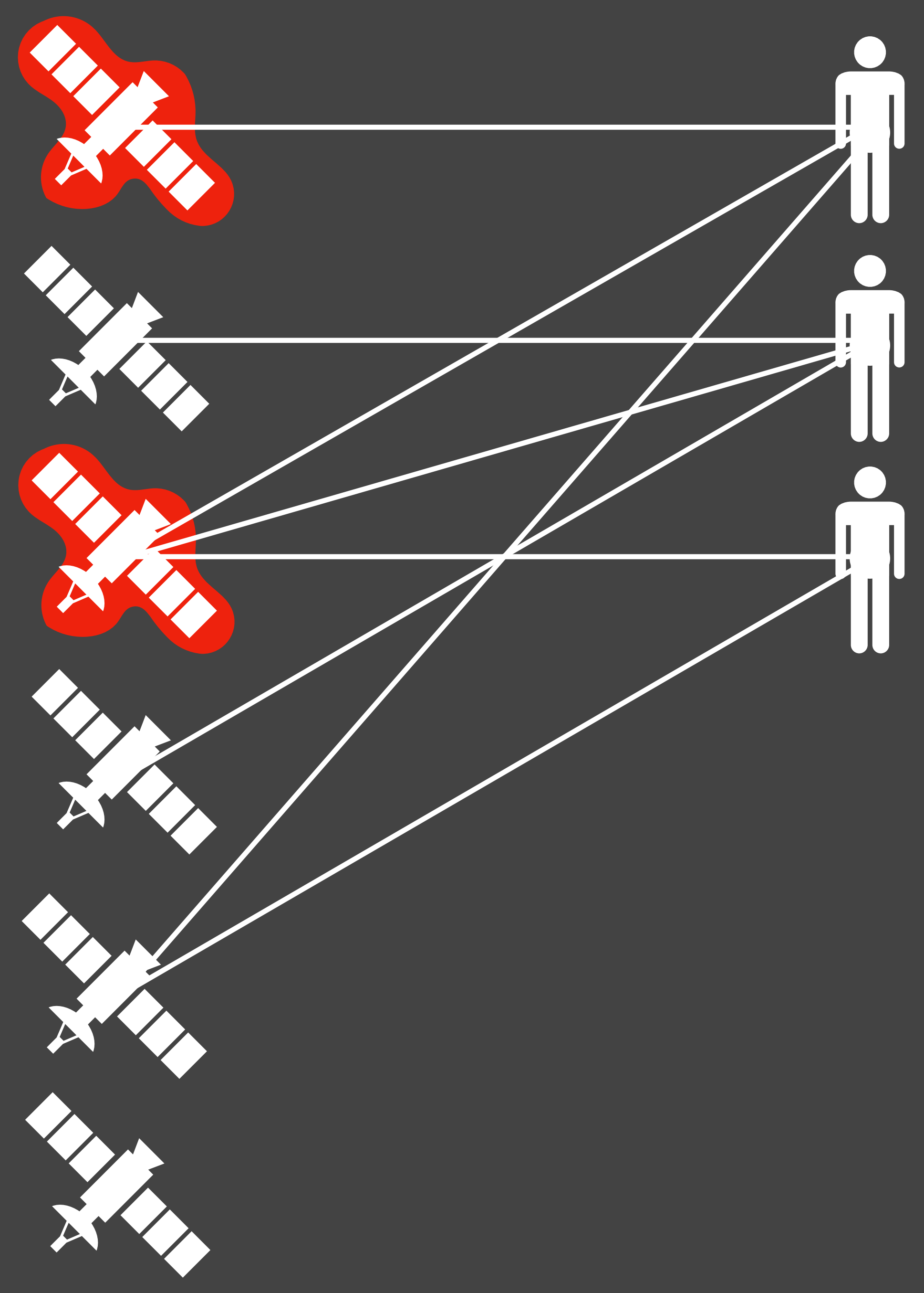
My Work

Online

Dynamic

Streaming

No take-backs



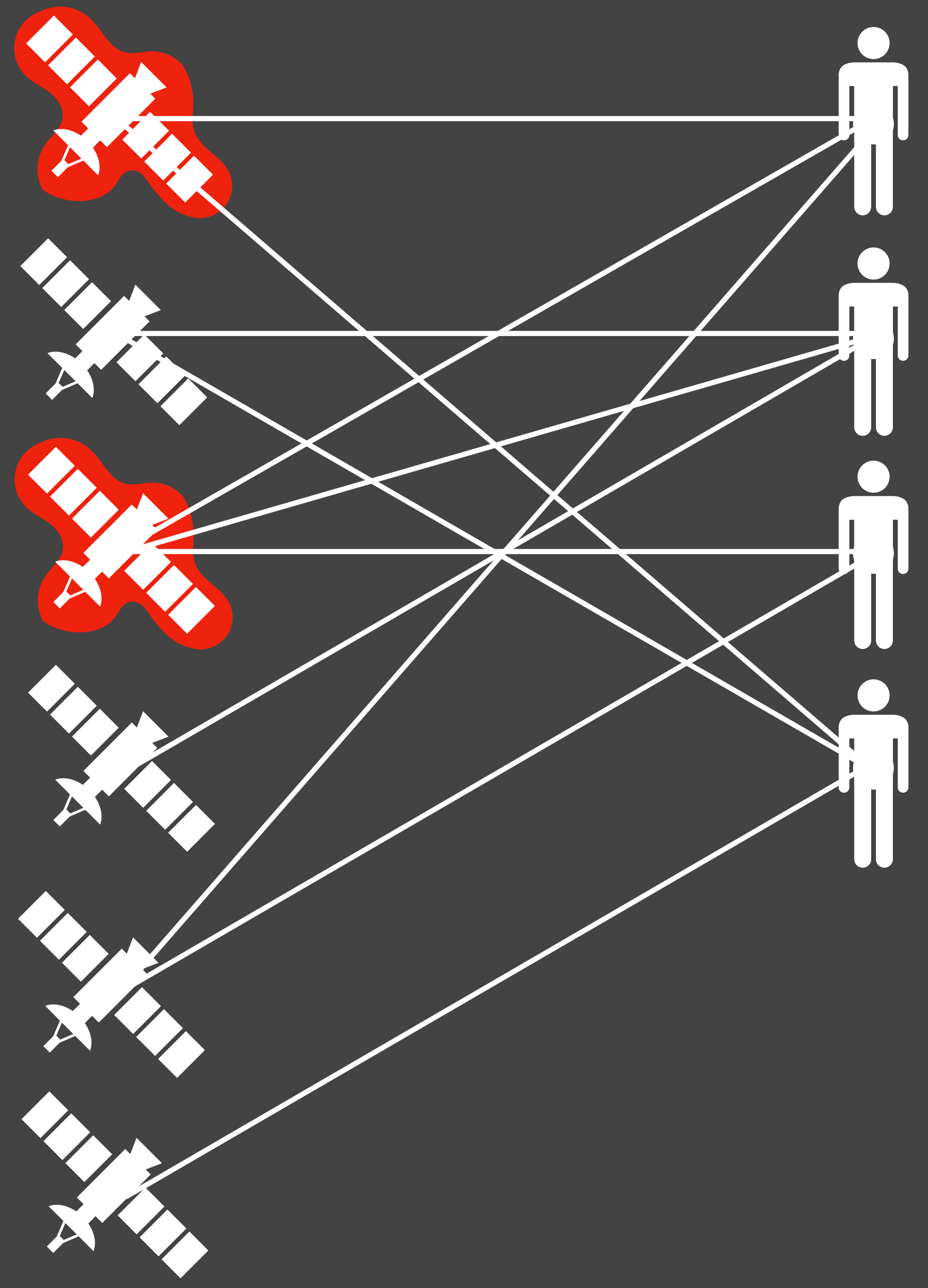
My Work

Online

Dynamic

Streaming

No take-backs



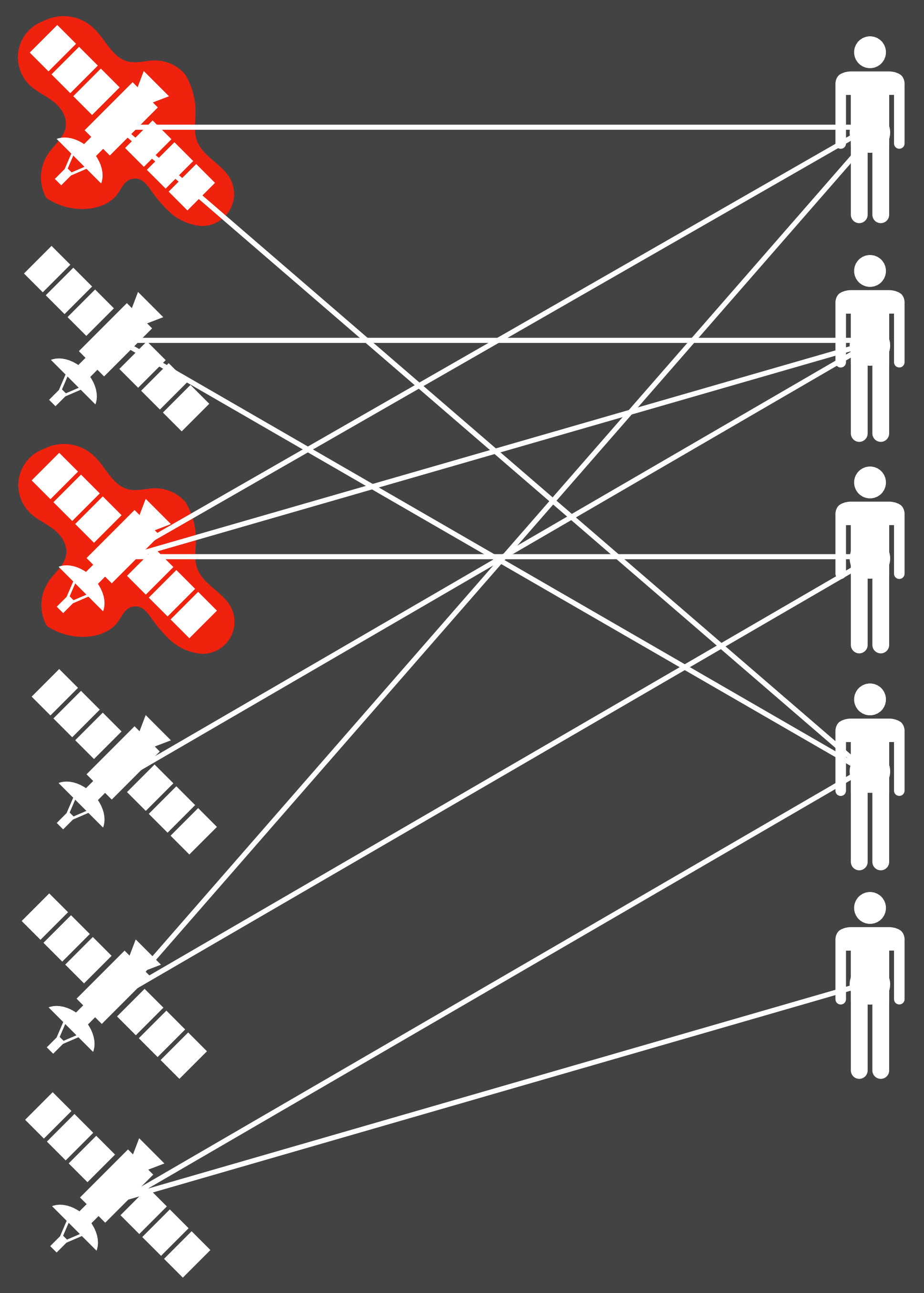
My Work

Online

Dynamic

Streaming

No take-backs



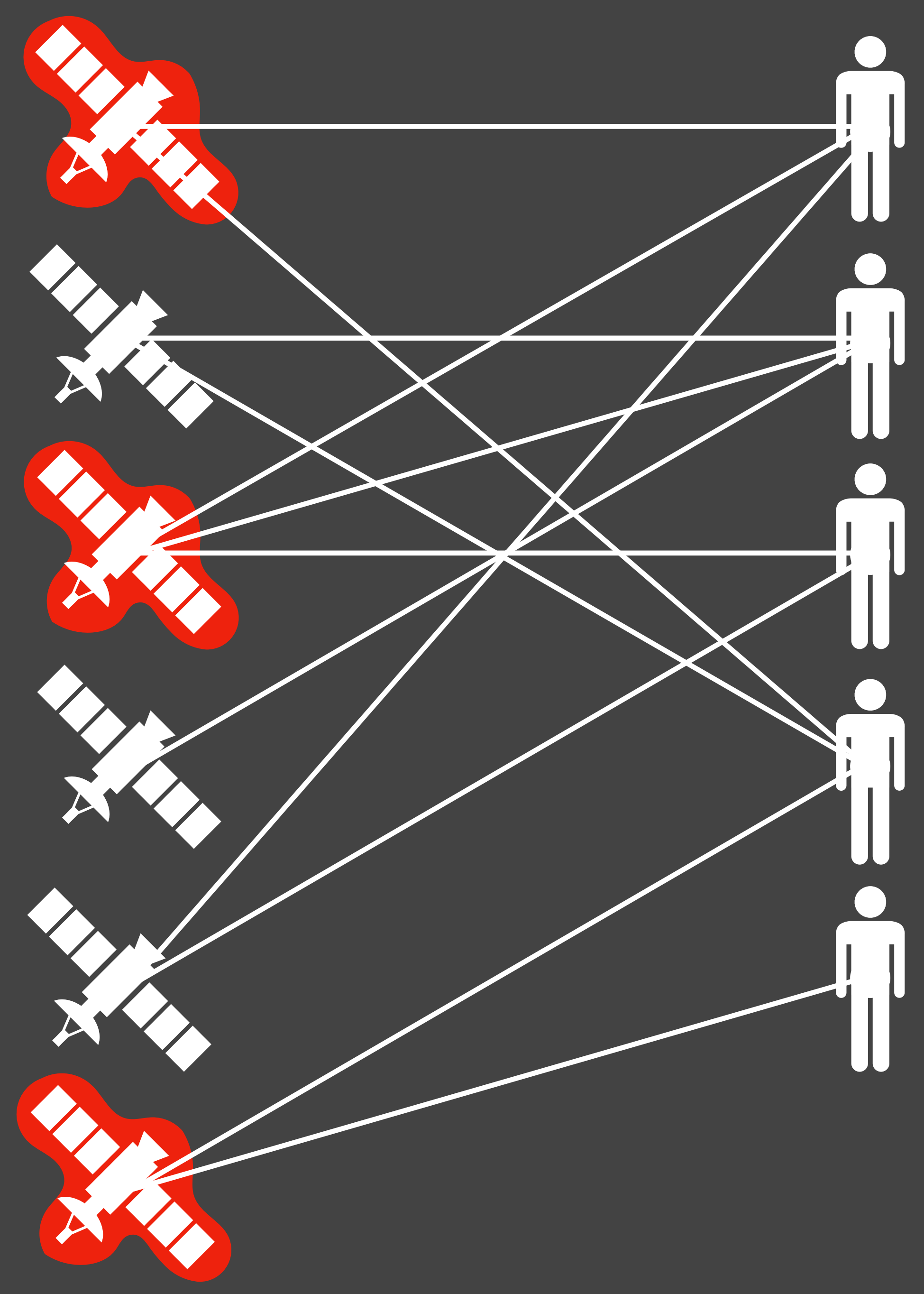
My Work

Online

Dynamic

Streaming

No take-backs



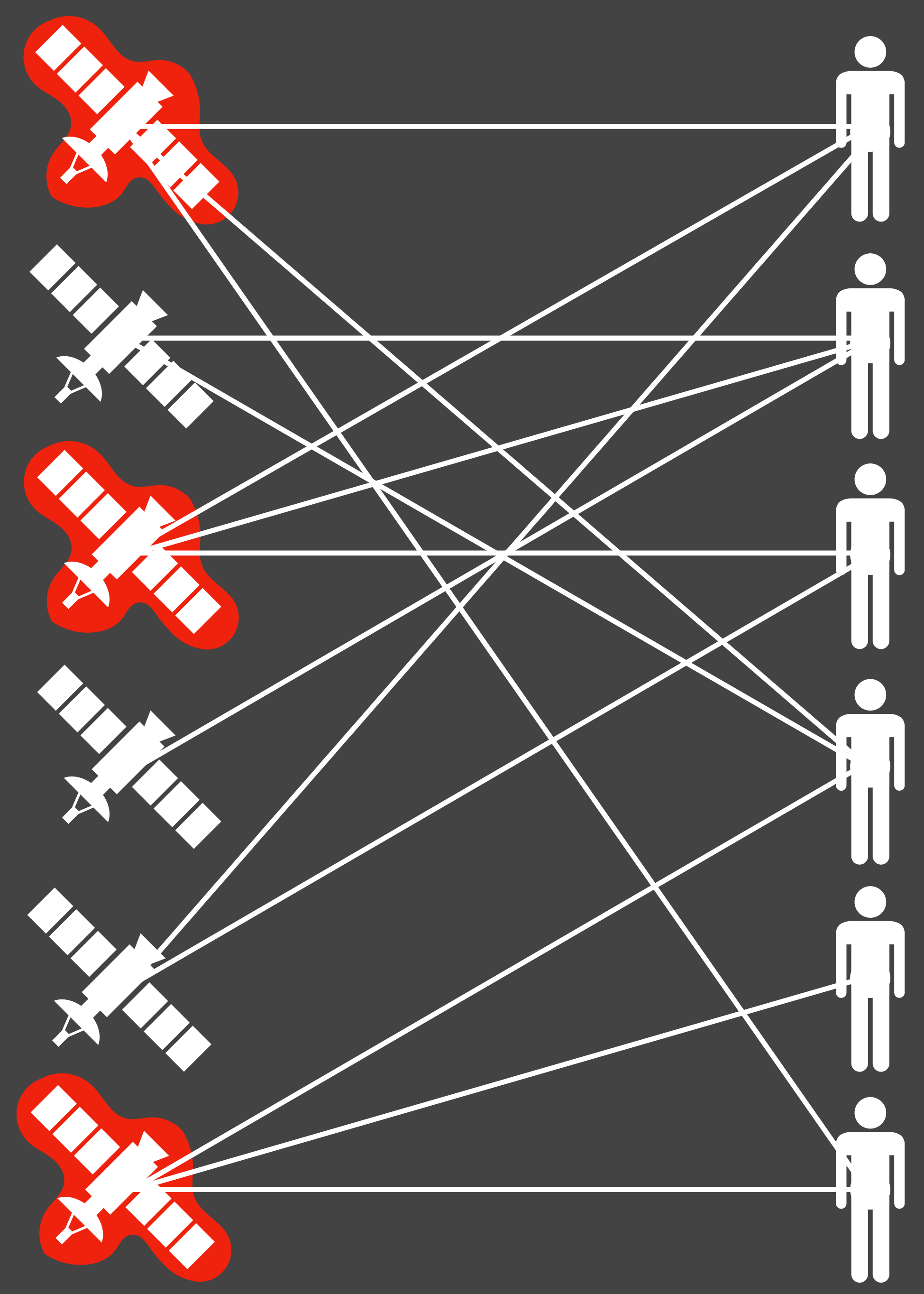
My Work

Online

Dynamic

Streaming

No take-backs



My Work

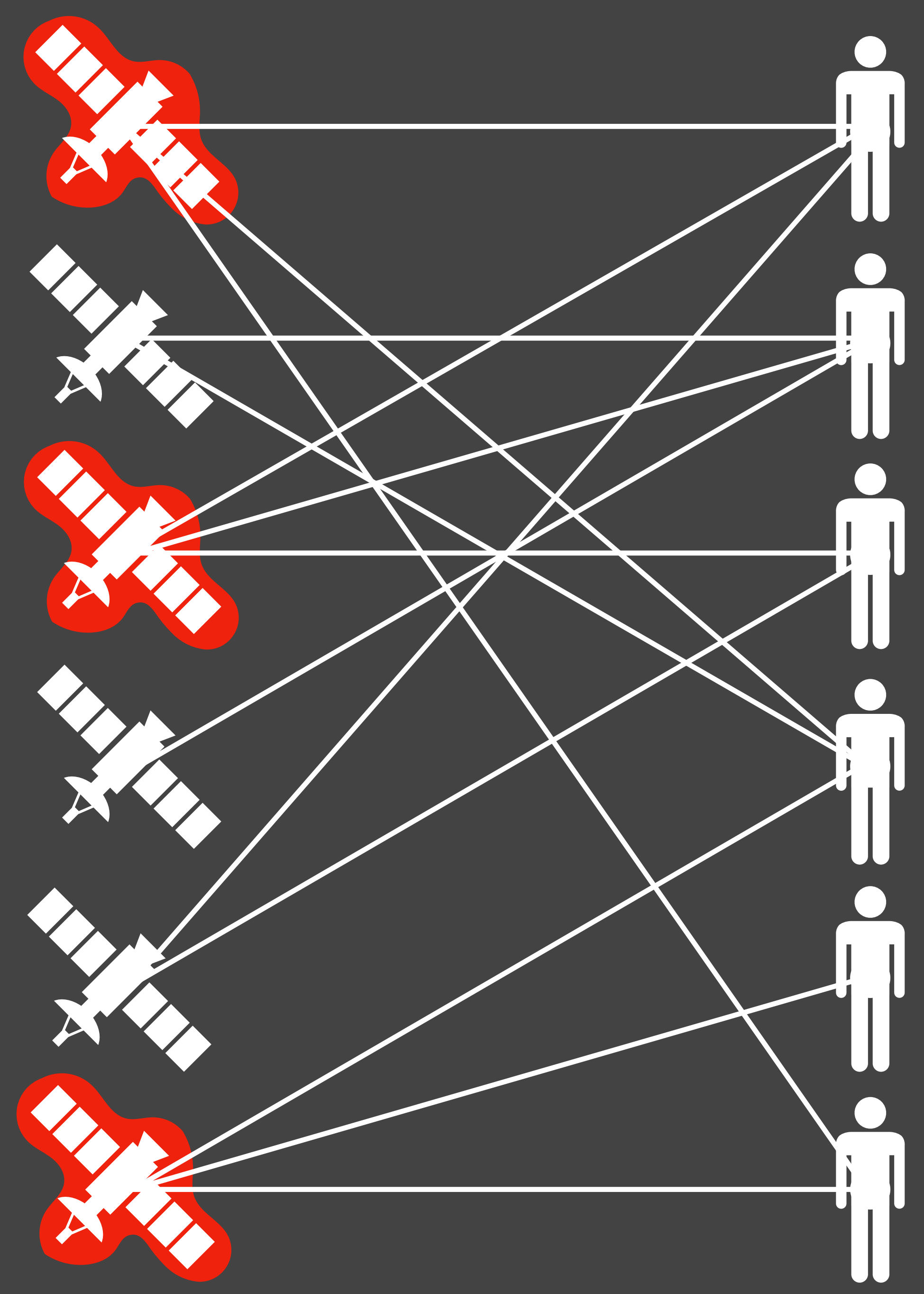
Online

Dynamic

Streaming

No take-backs

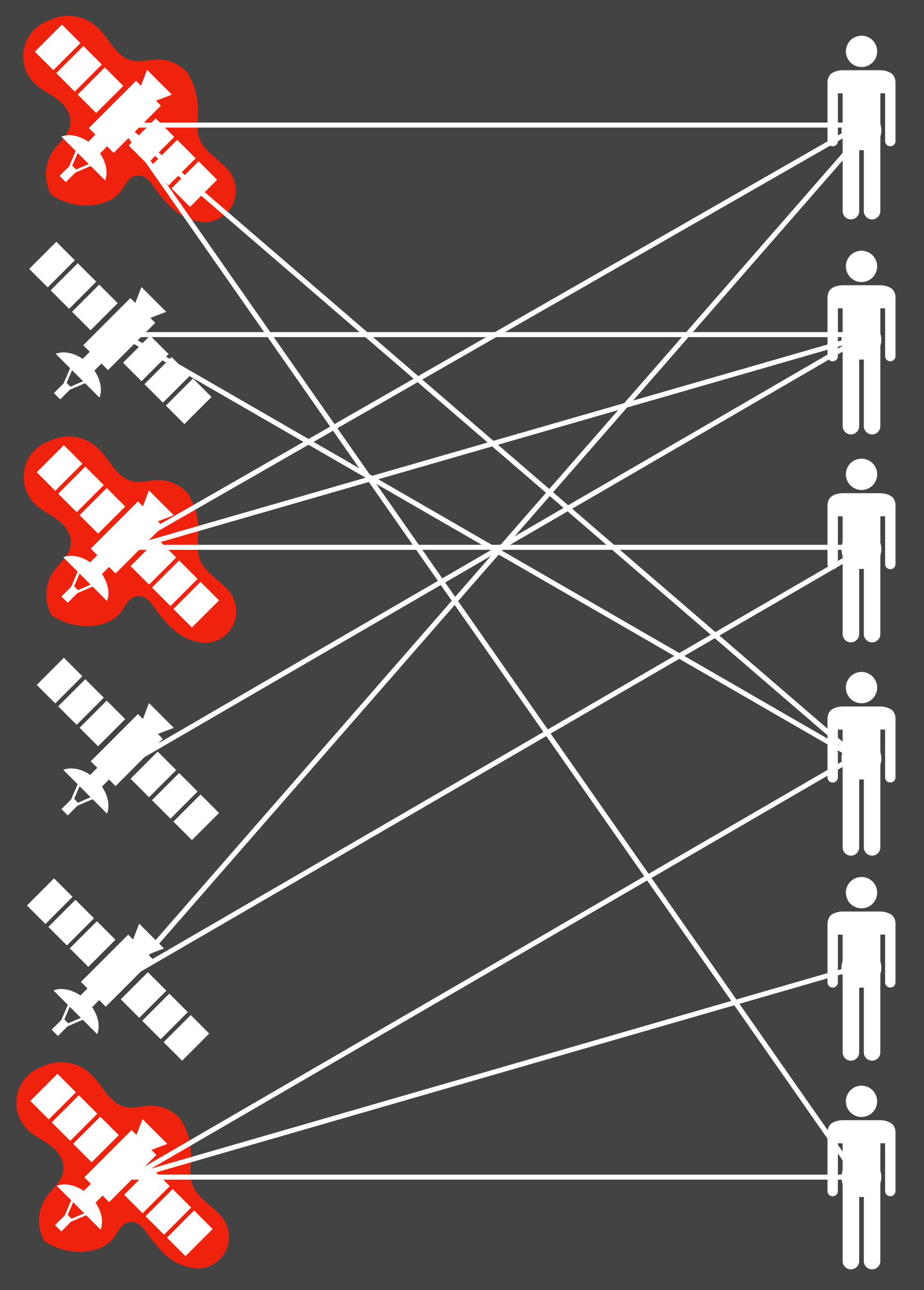
Low movement



My Work

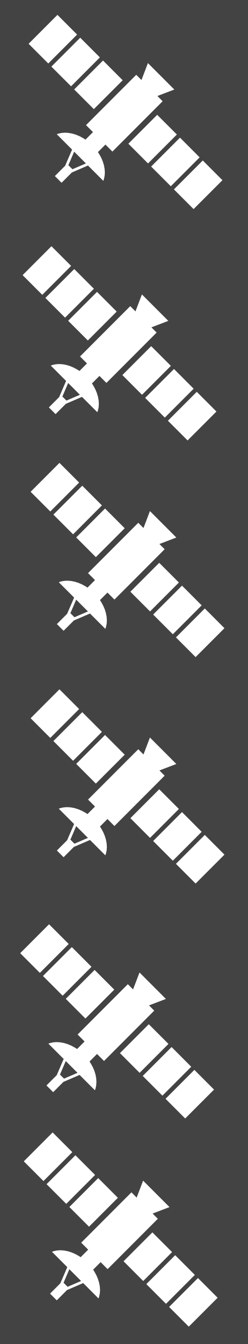
Online

No take-backs



Dynamic

Low movement

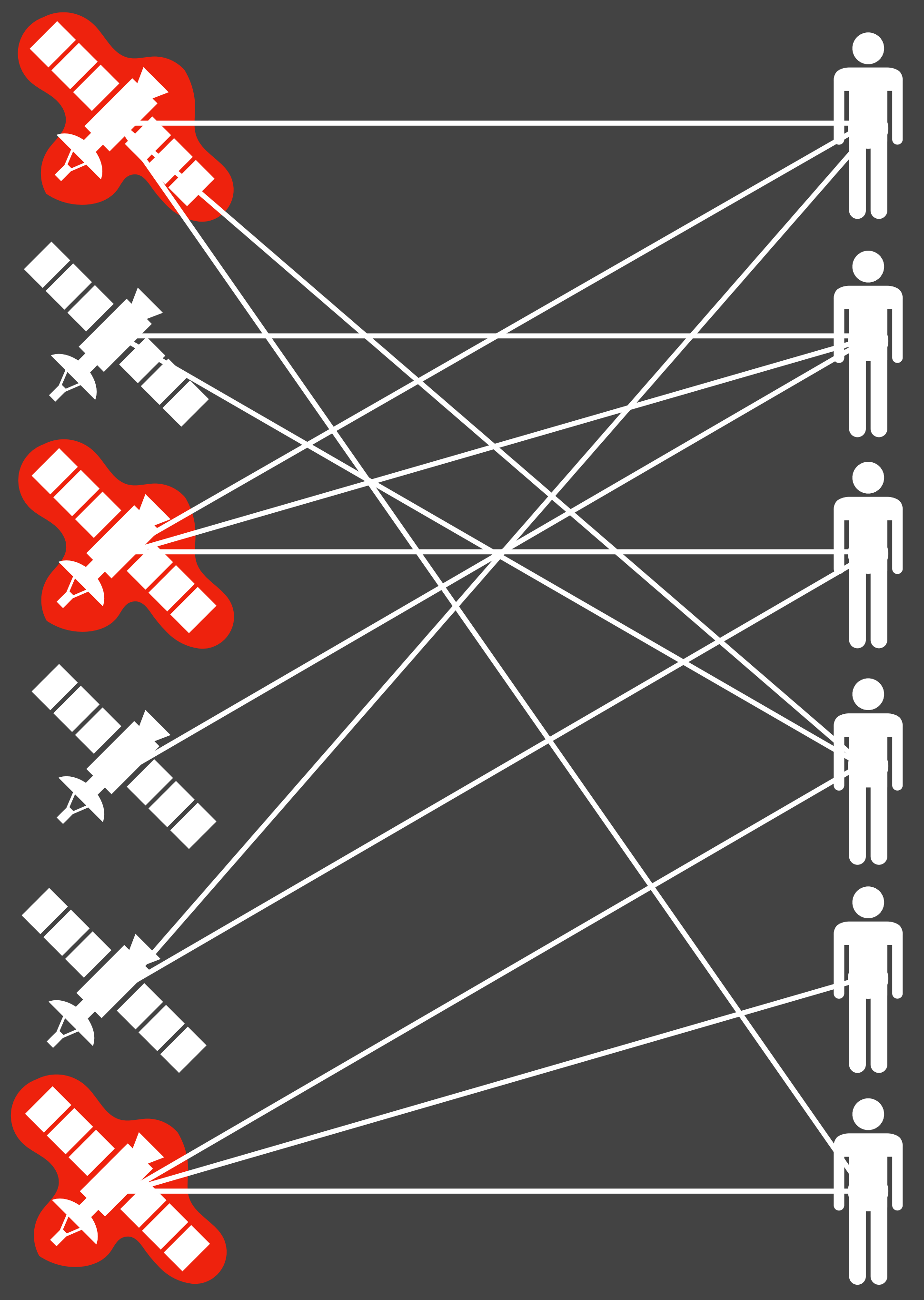


Streaming

My Work

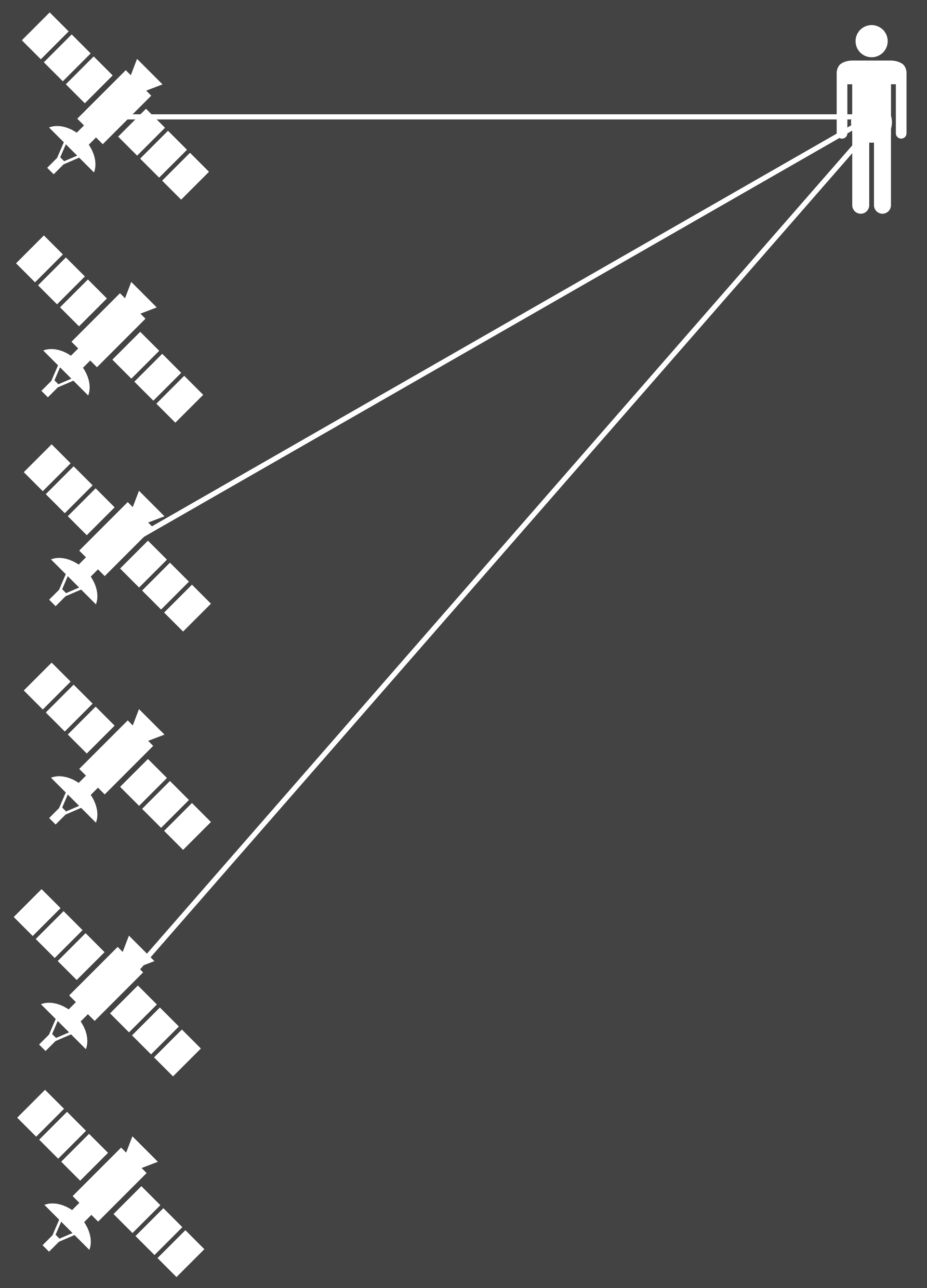
Online

No take-backs



Dynamic

Low movement

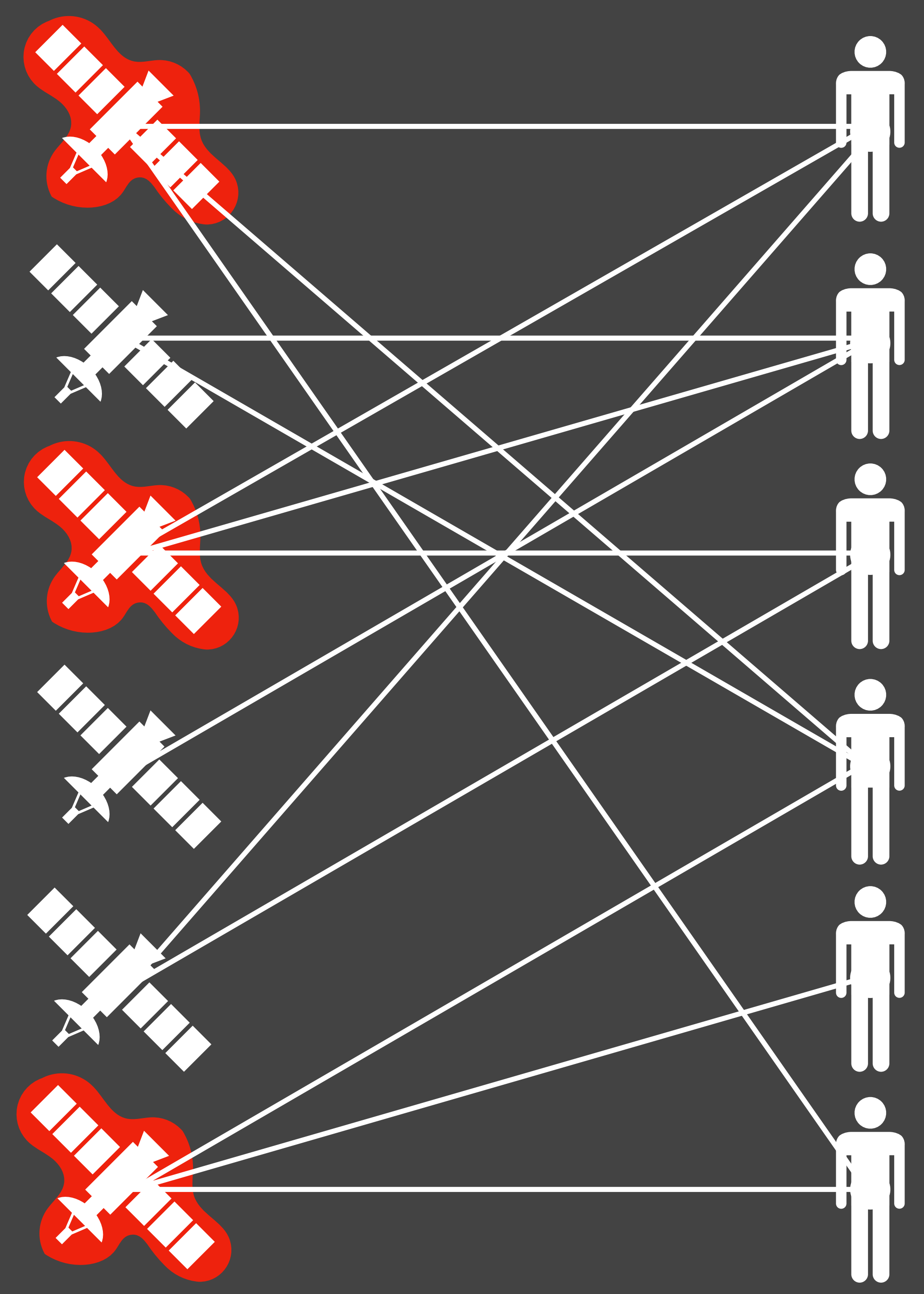


Streaming

My Work

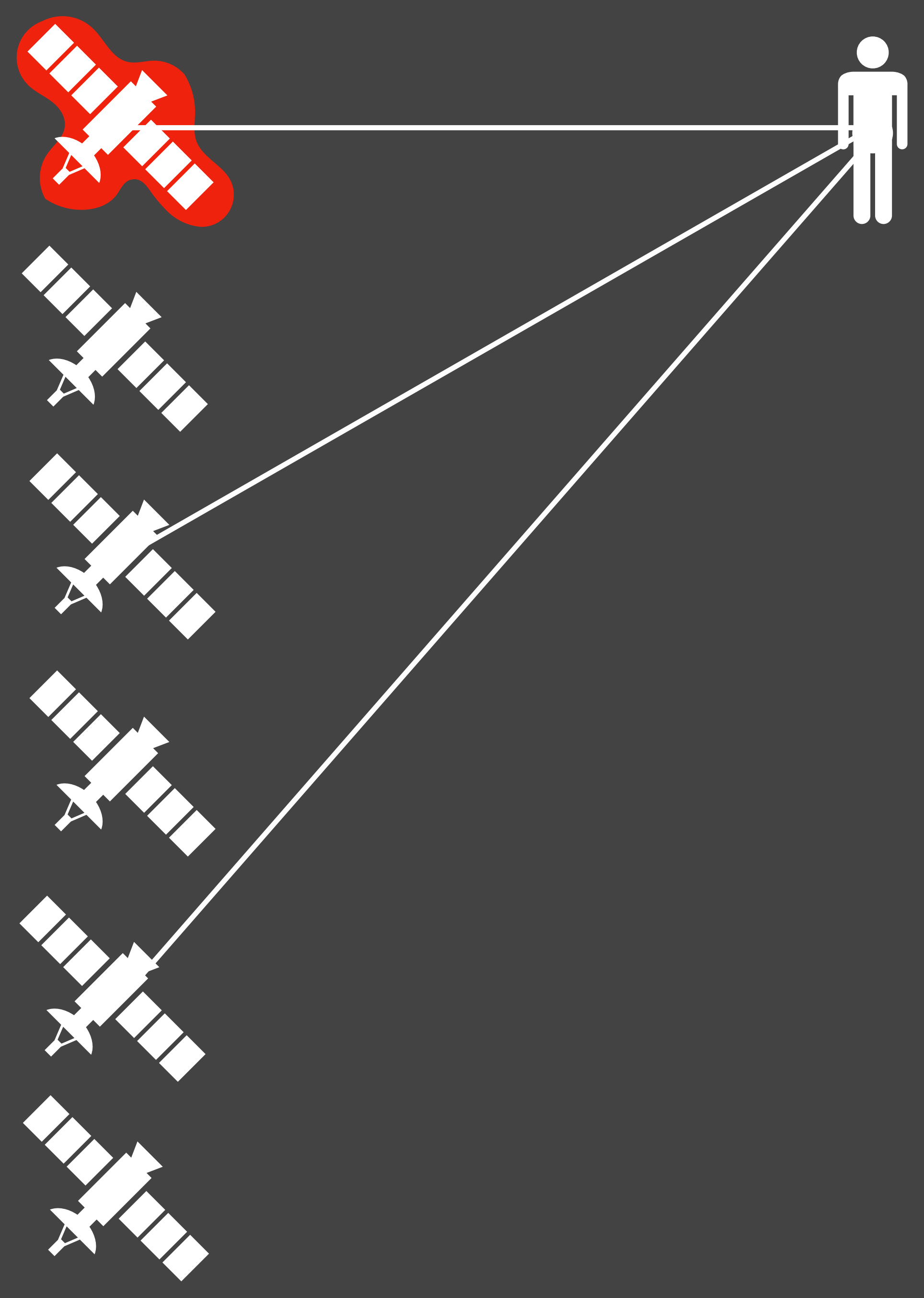
Online

No take-backs



Dynamic

Low movement



Streaming

My Work

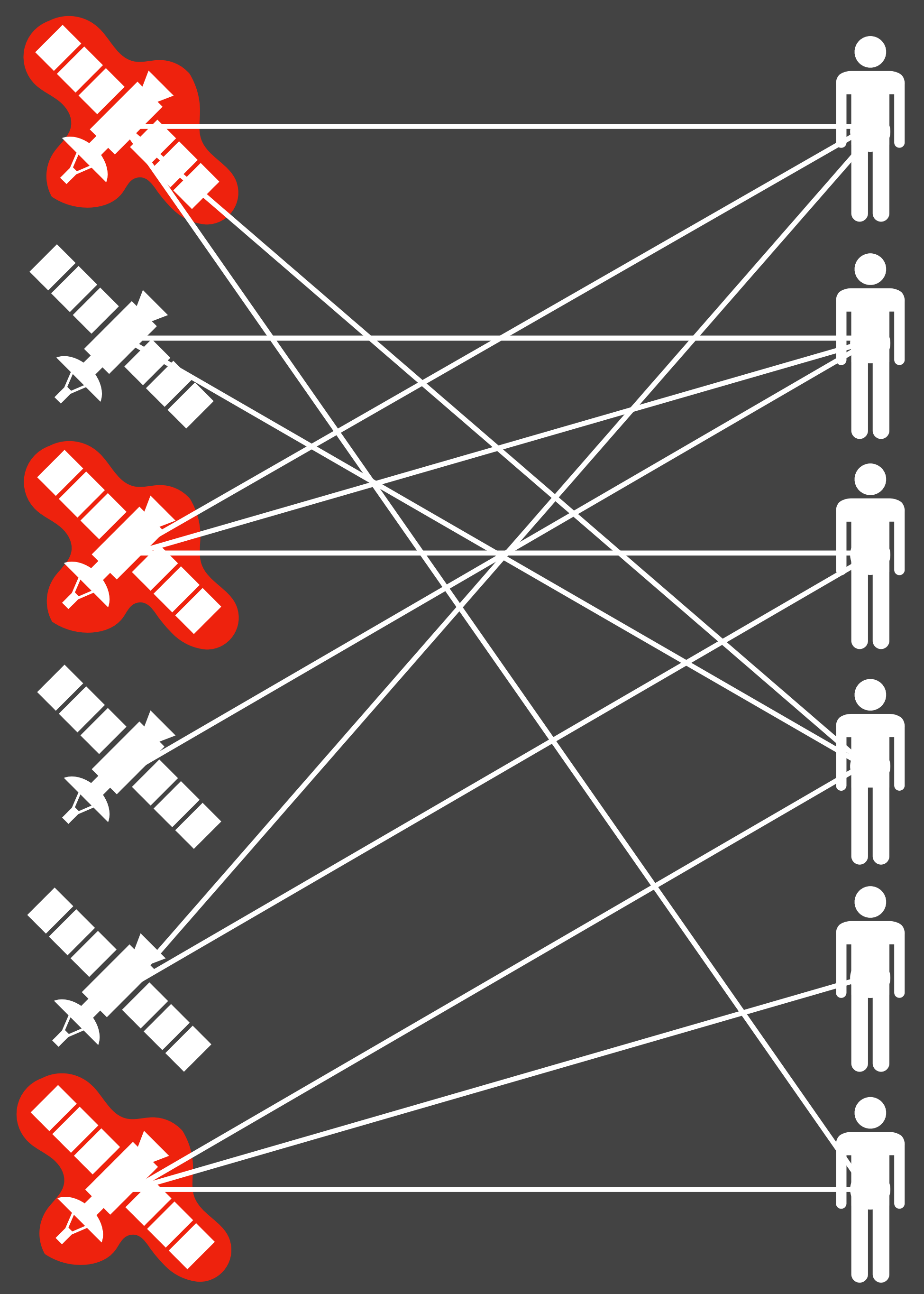
Online

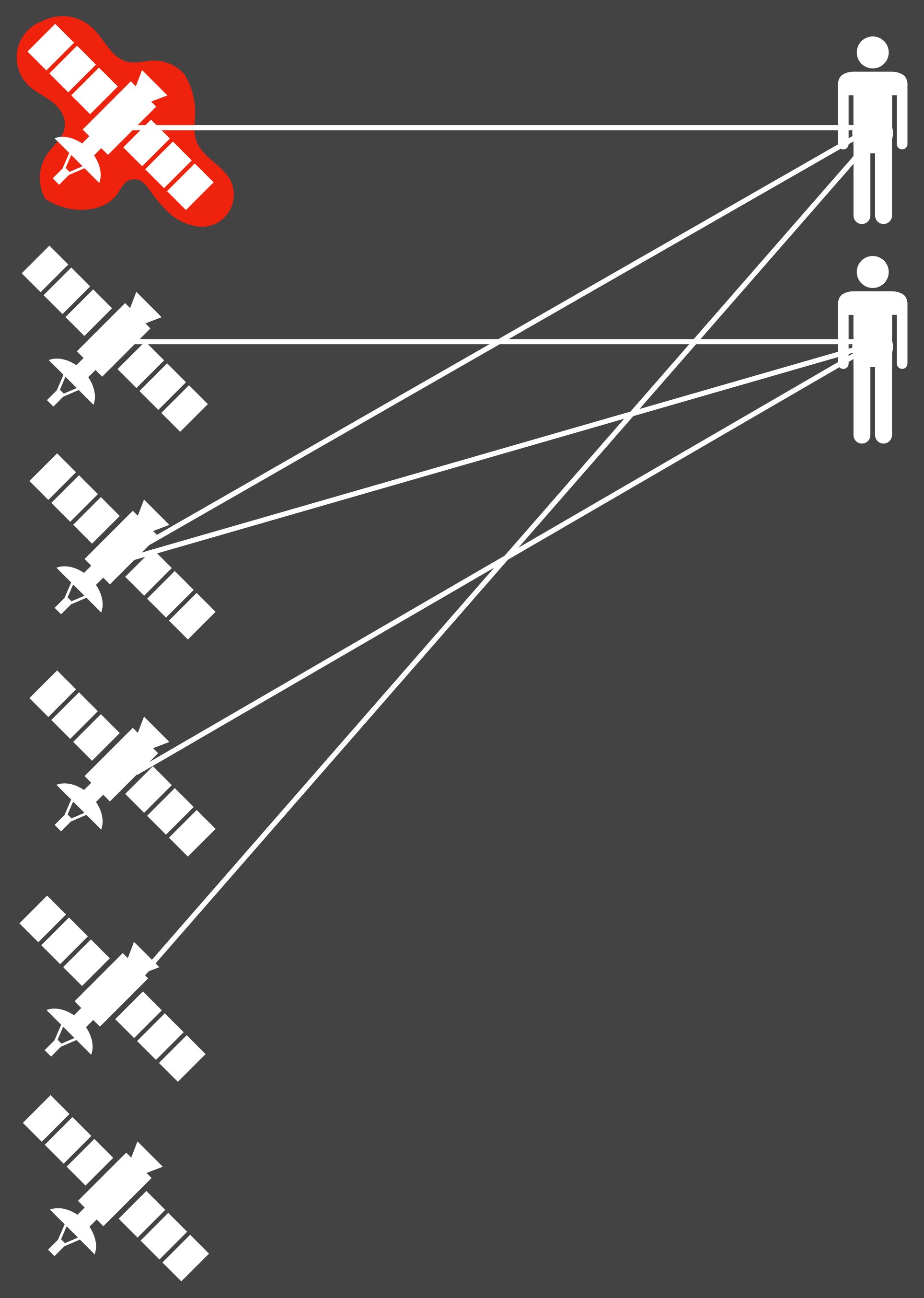
Dynamic

Streaming

No take-backs

Low movement

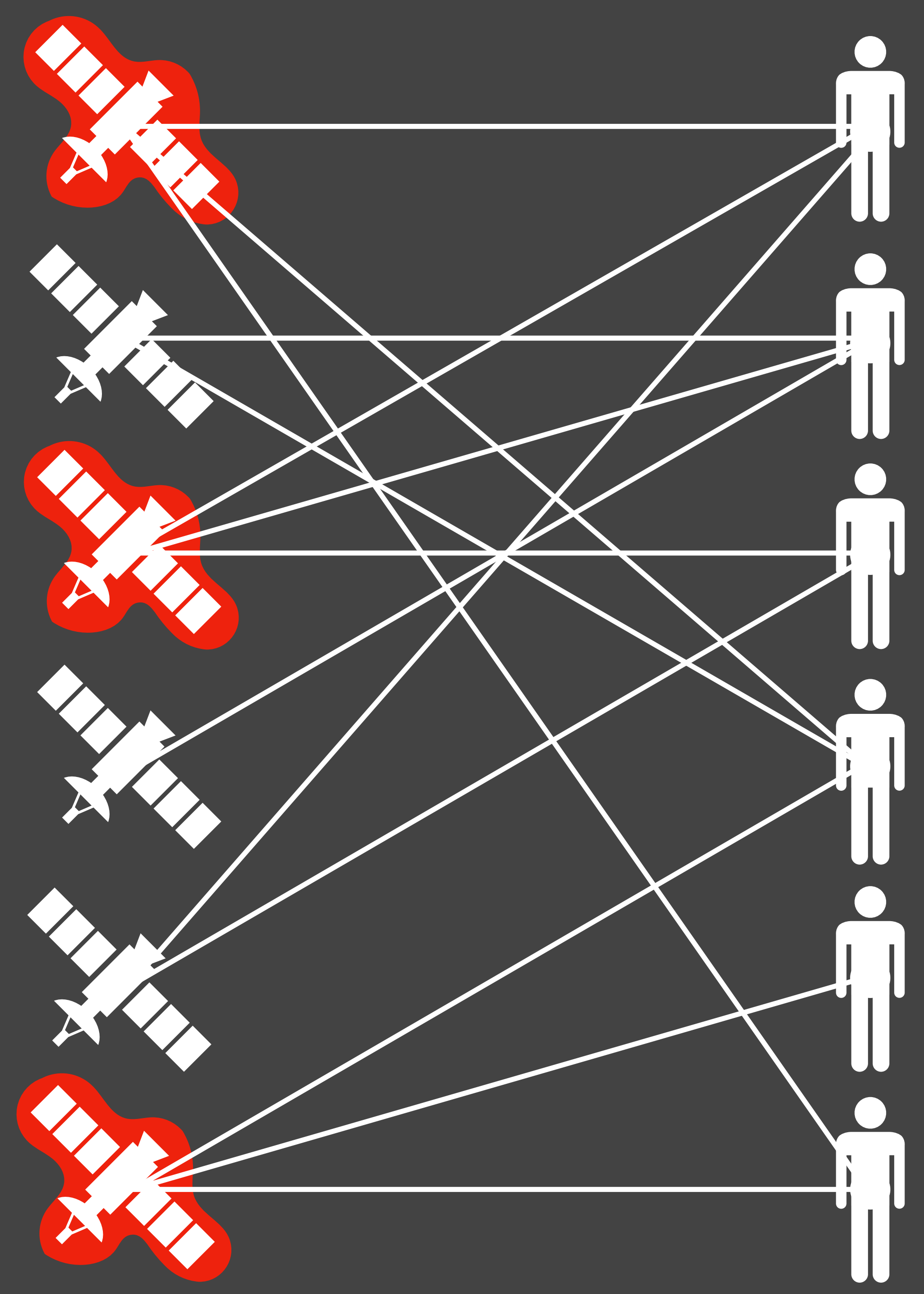




My Work

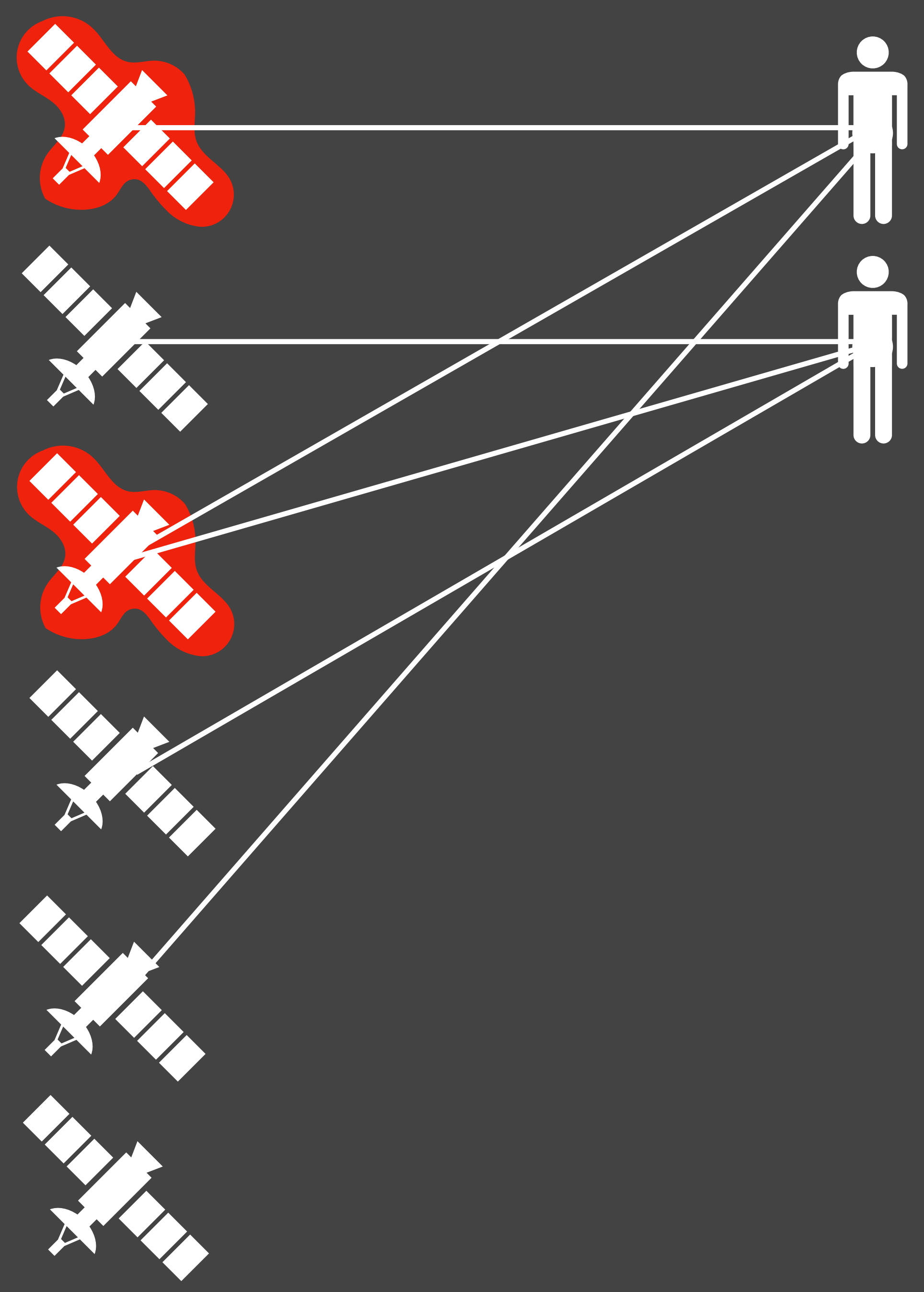
Online

No take-backs



Dynamic

Low movement

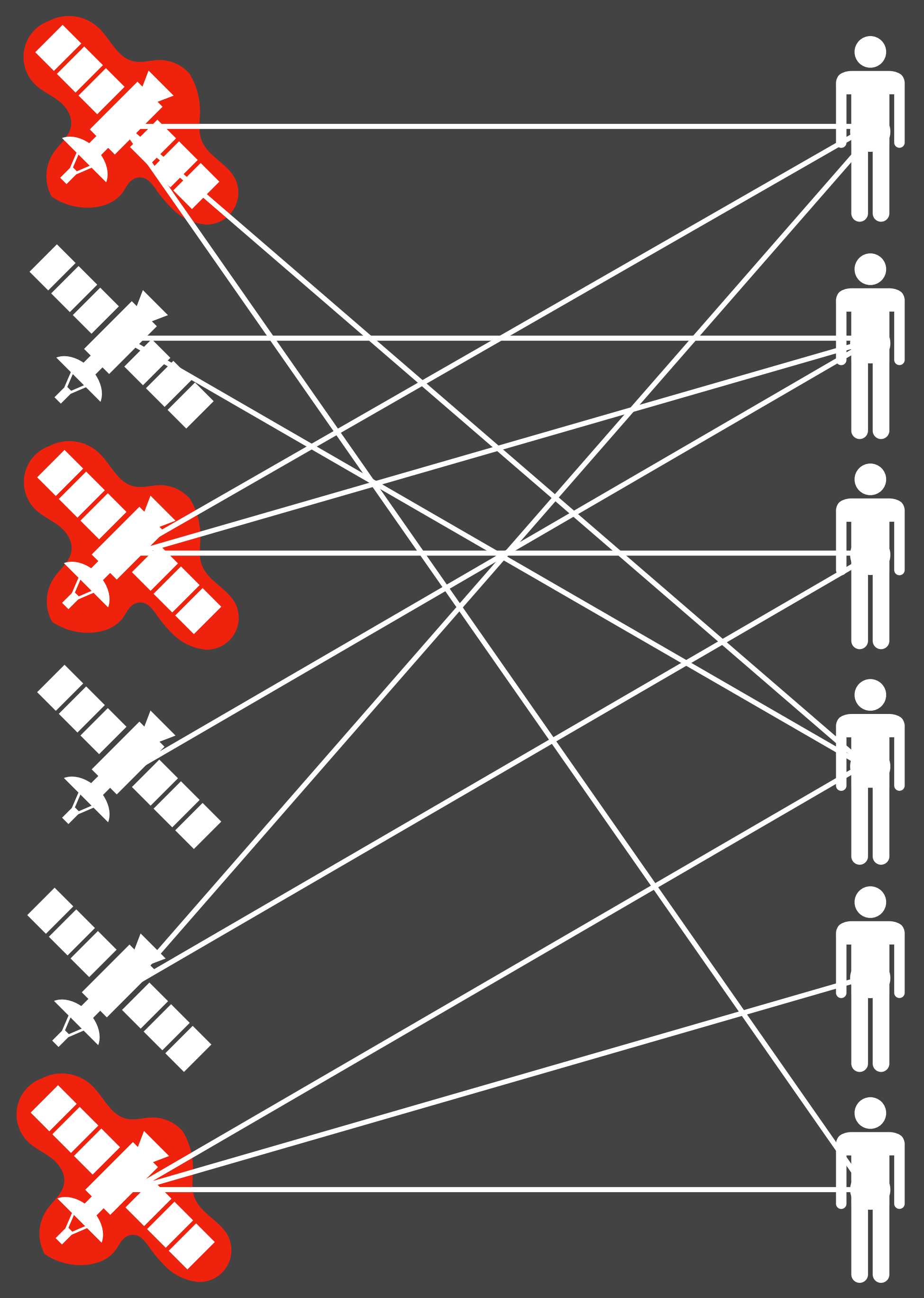


Streaming

My Work

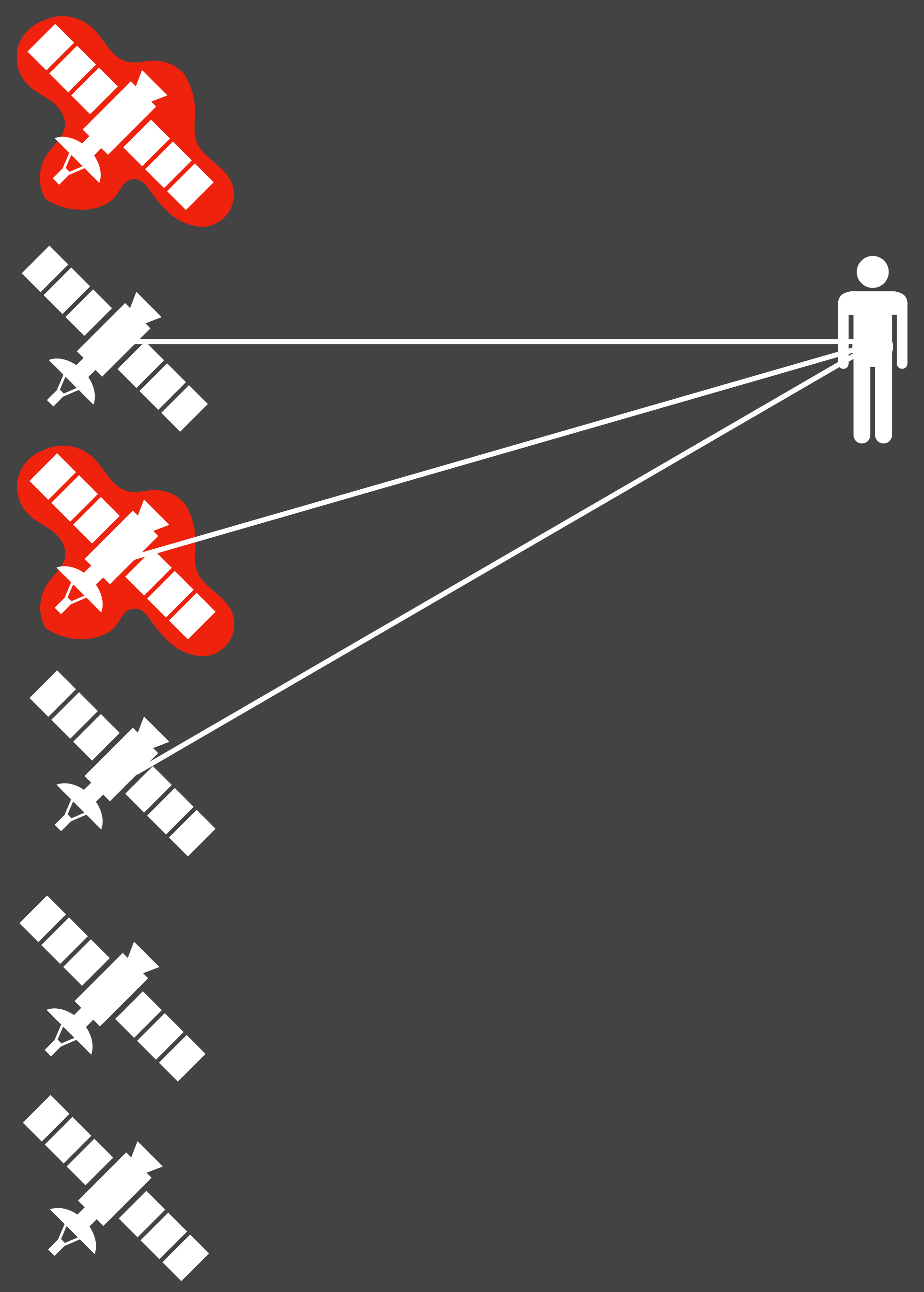
Online

No take-backs



Dynamic

Low movement

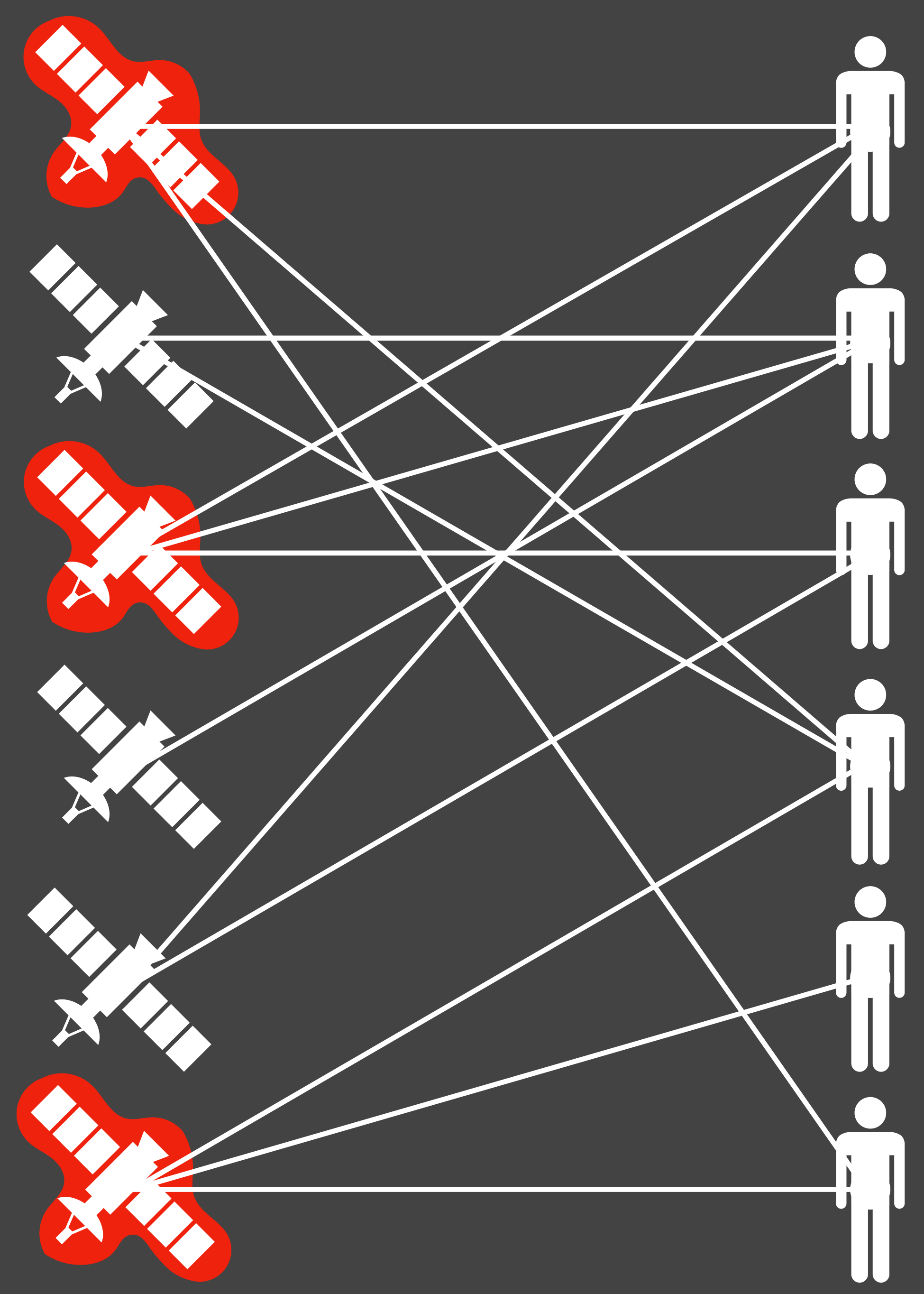


Streaming

My Work

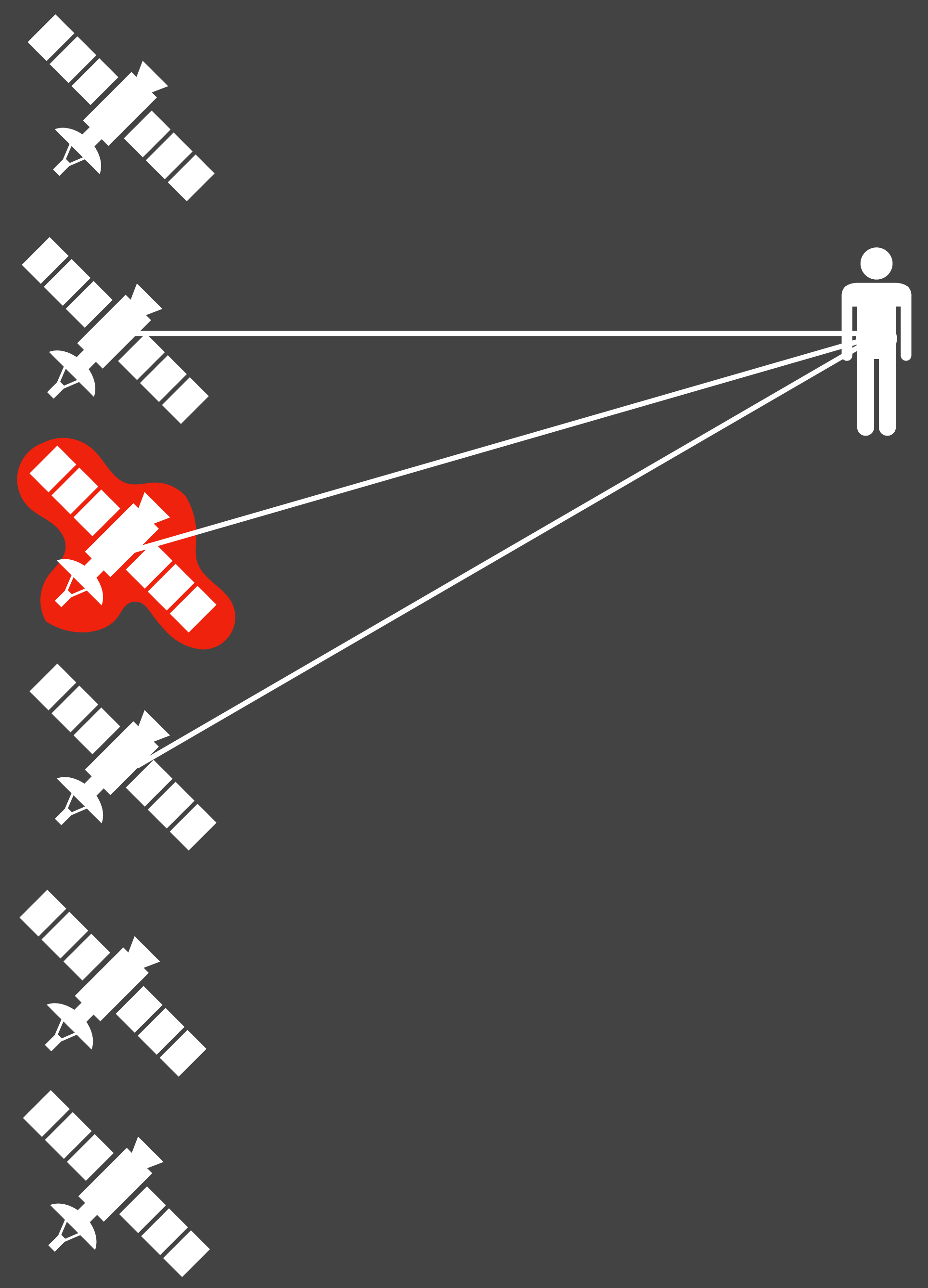
Online

No take-backs



Dynamic

Low movement

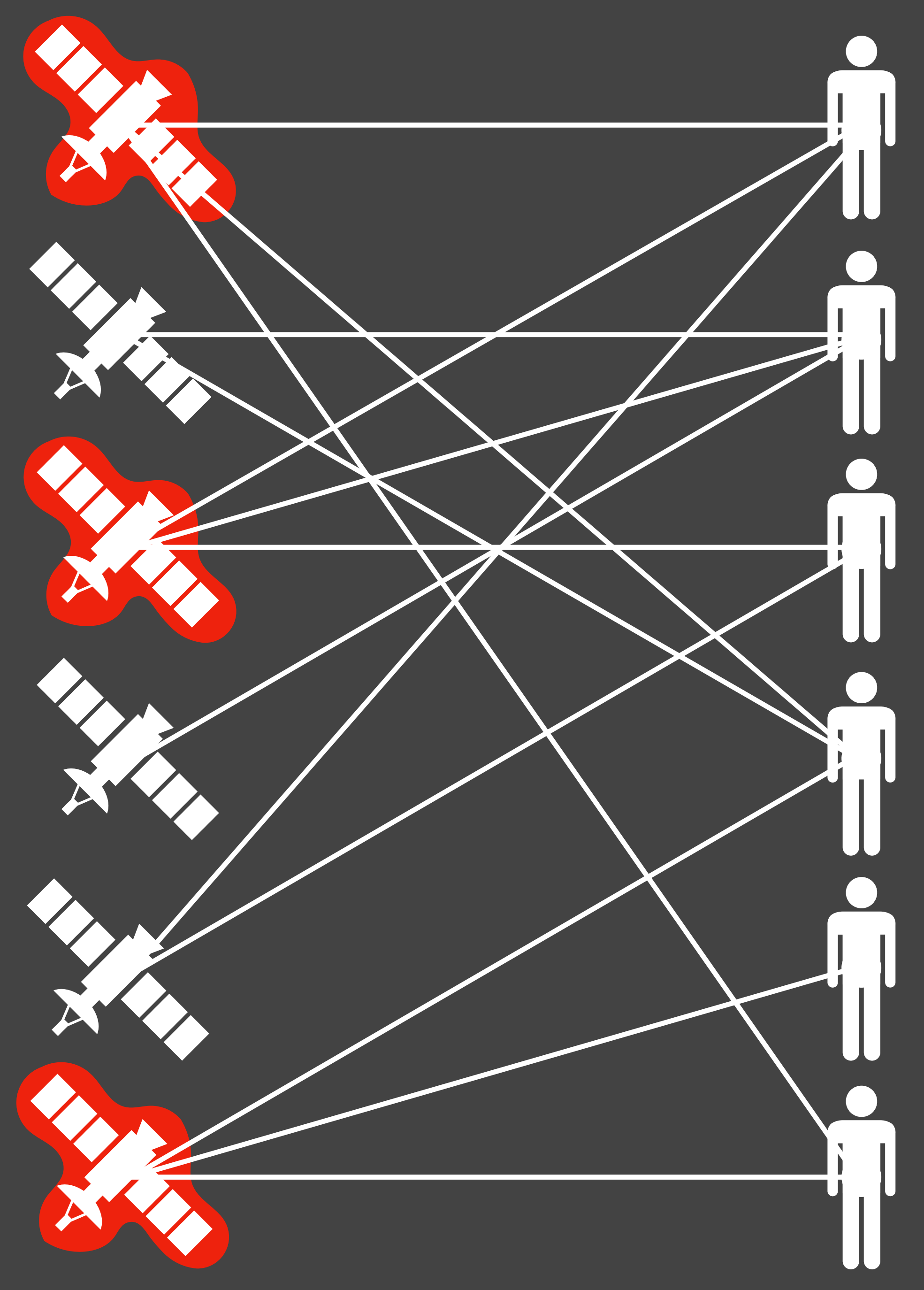


Streaming

My Work

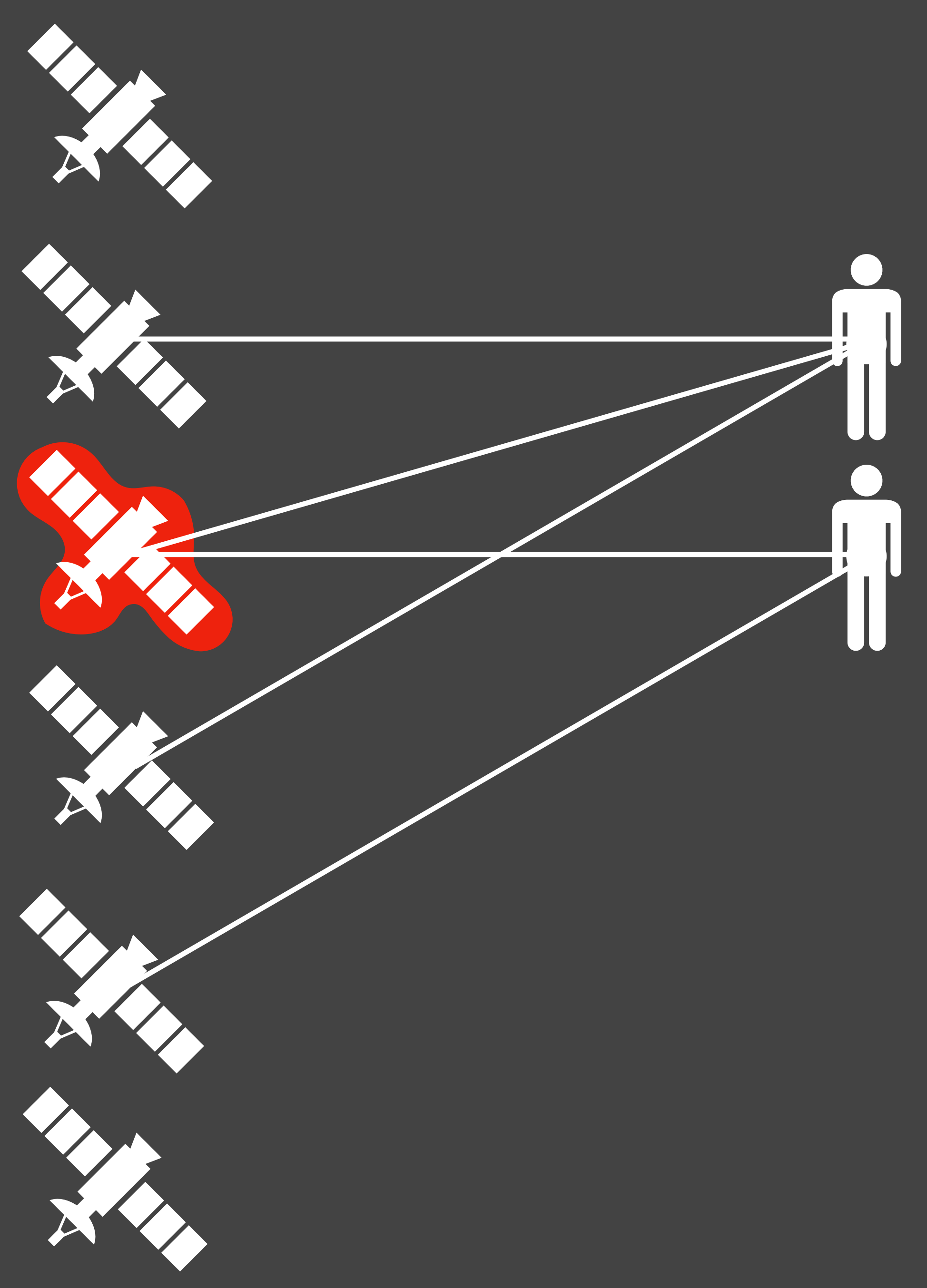
Online

No take-backs



Dynamic

Low movement

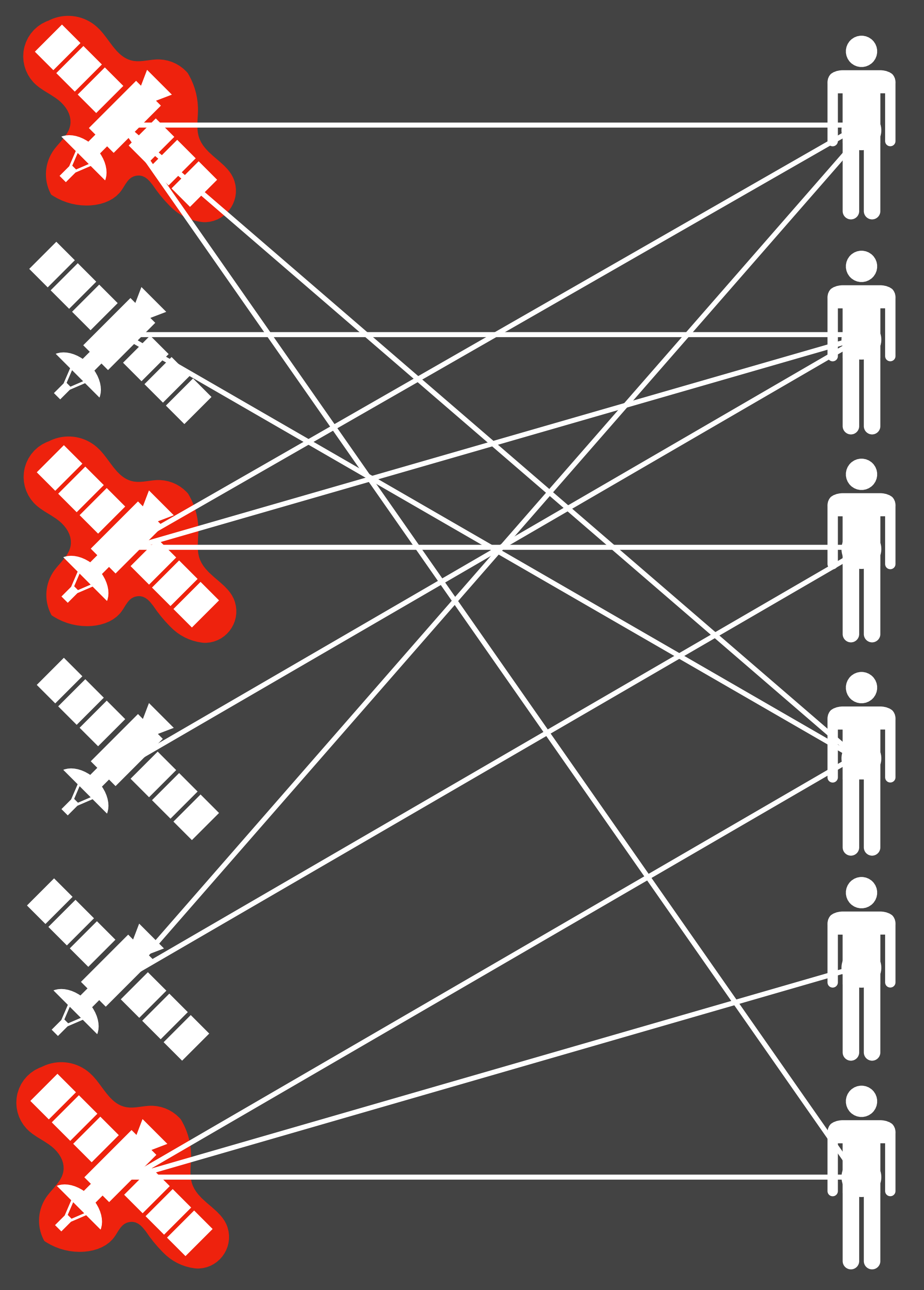


Streaming

My Work

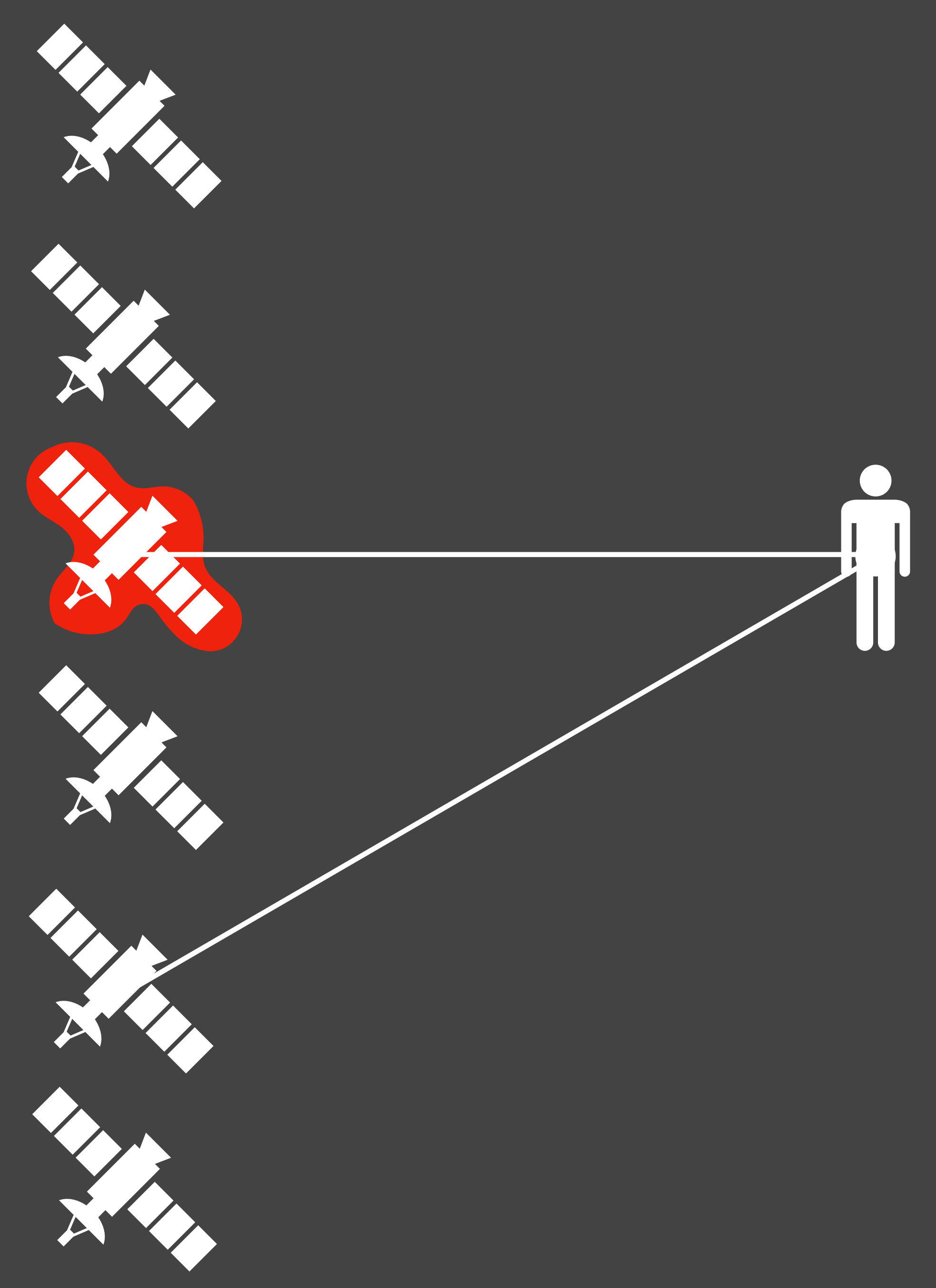
Online

No take-backs



Dynamic

Low movement



Streaming

My Work

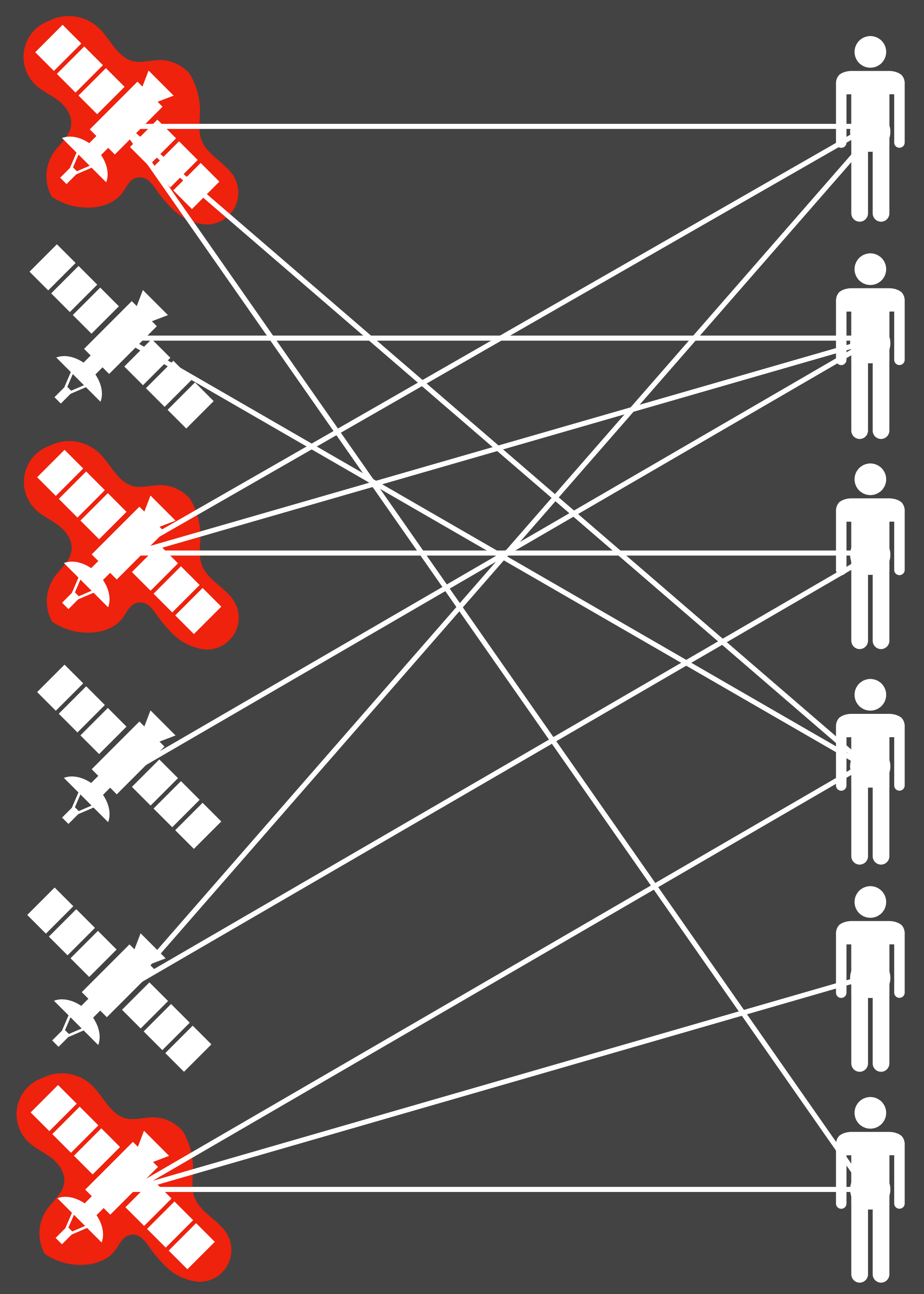
Online

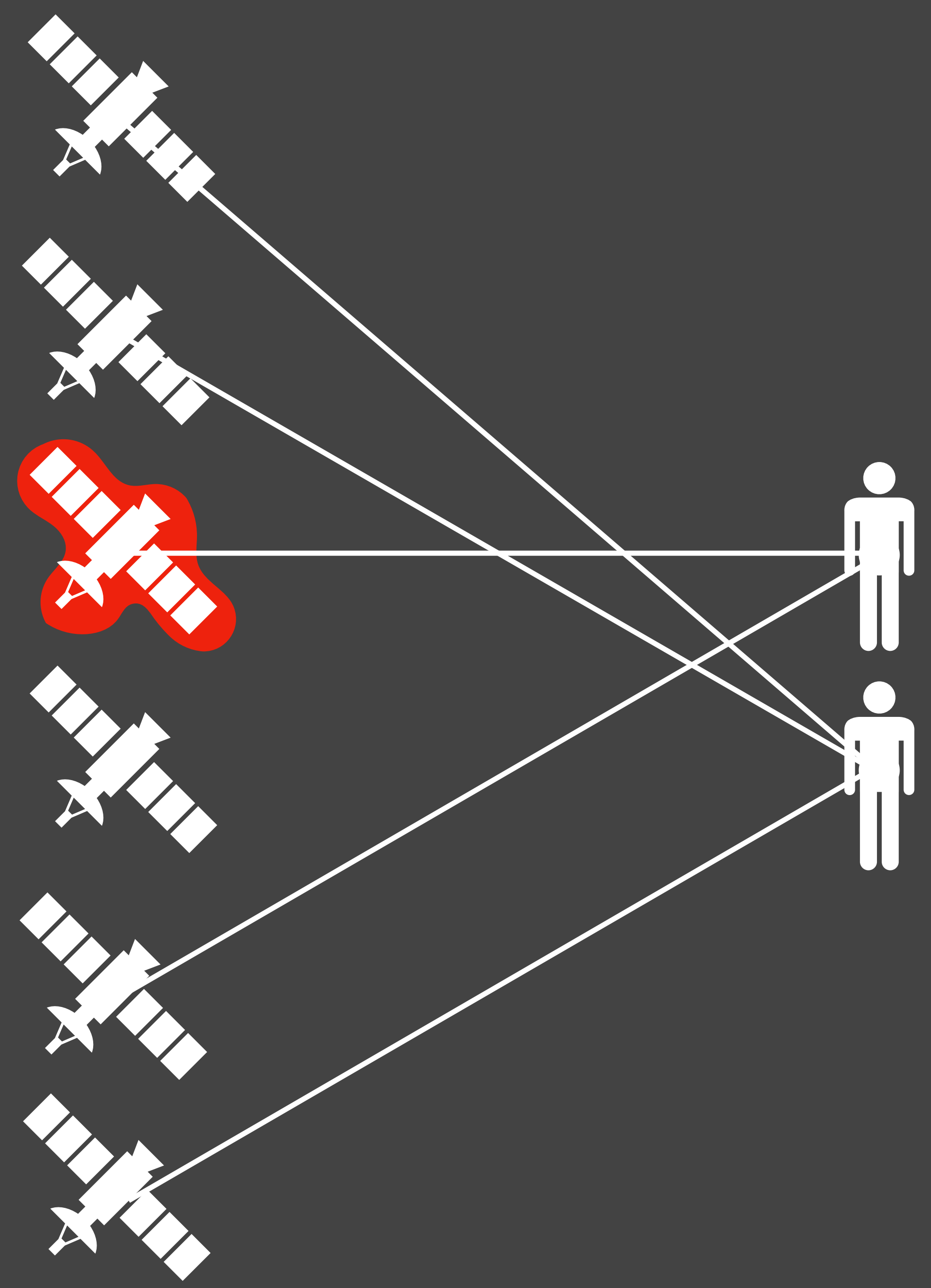
Dynamic

Streaming

No take-backs

Low movement

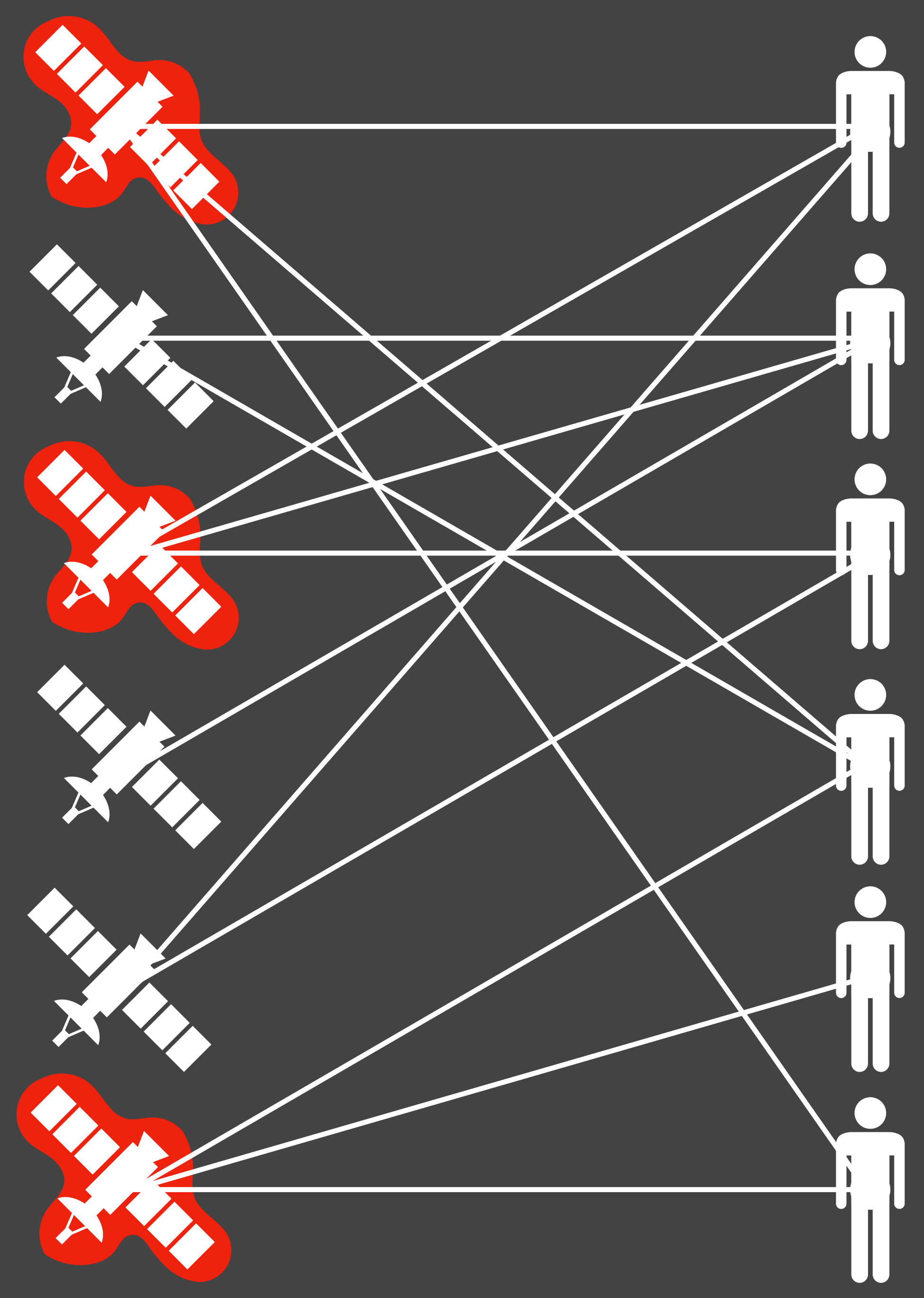




My Work

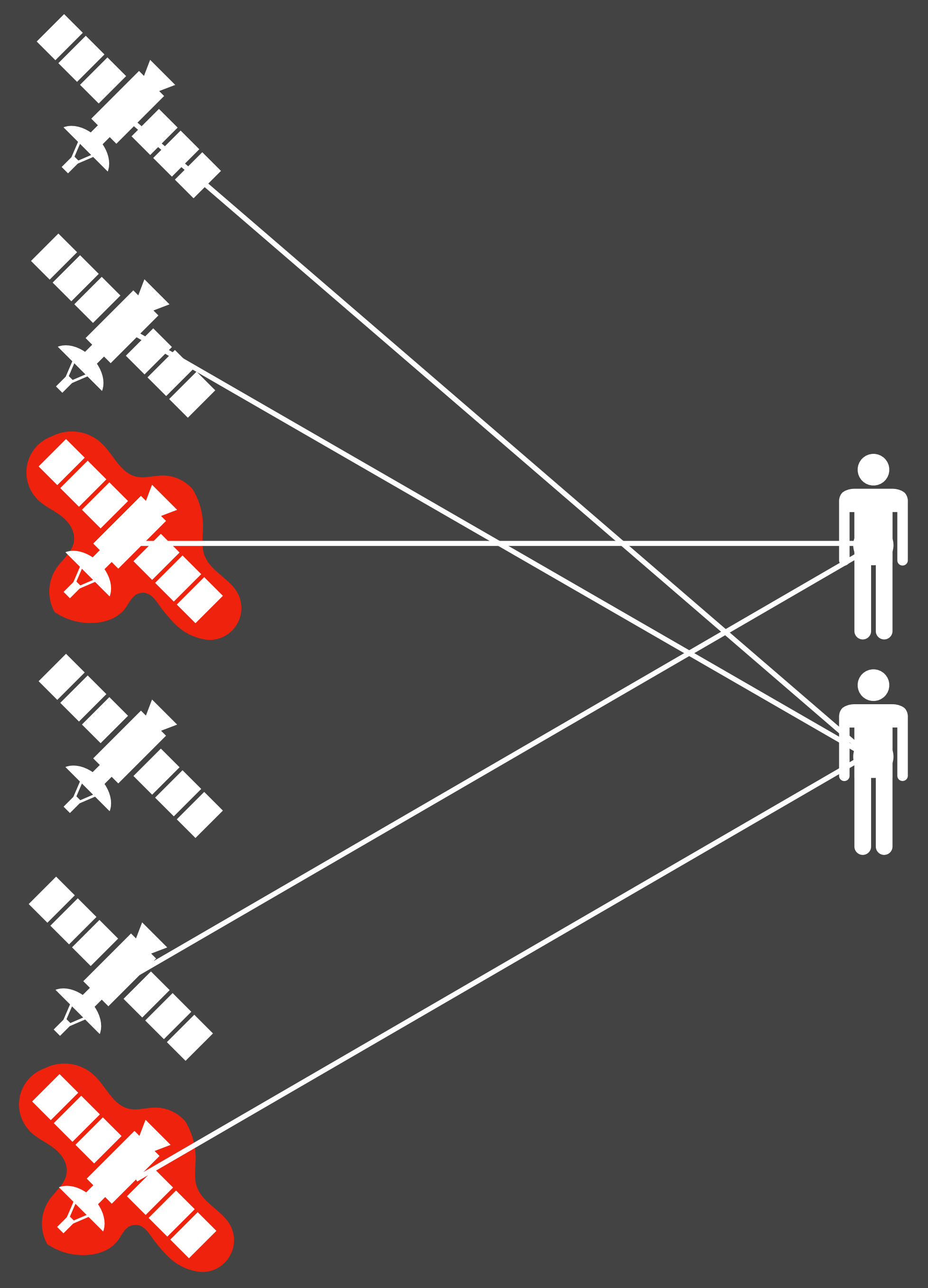
Online

No take-backs



Dynamic

Low movement

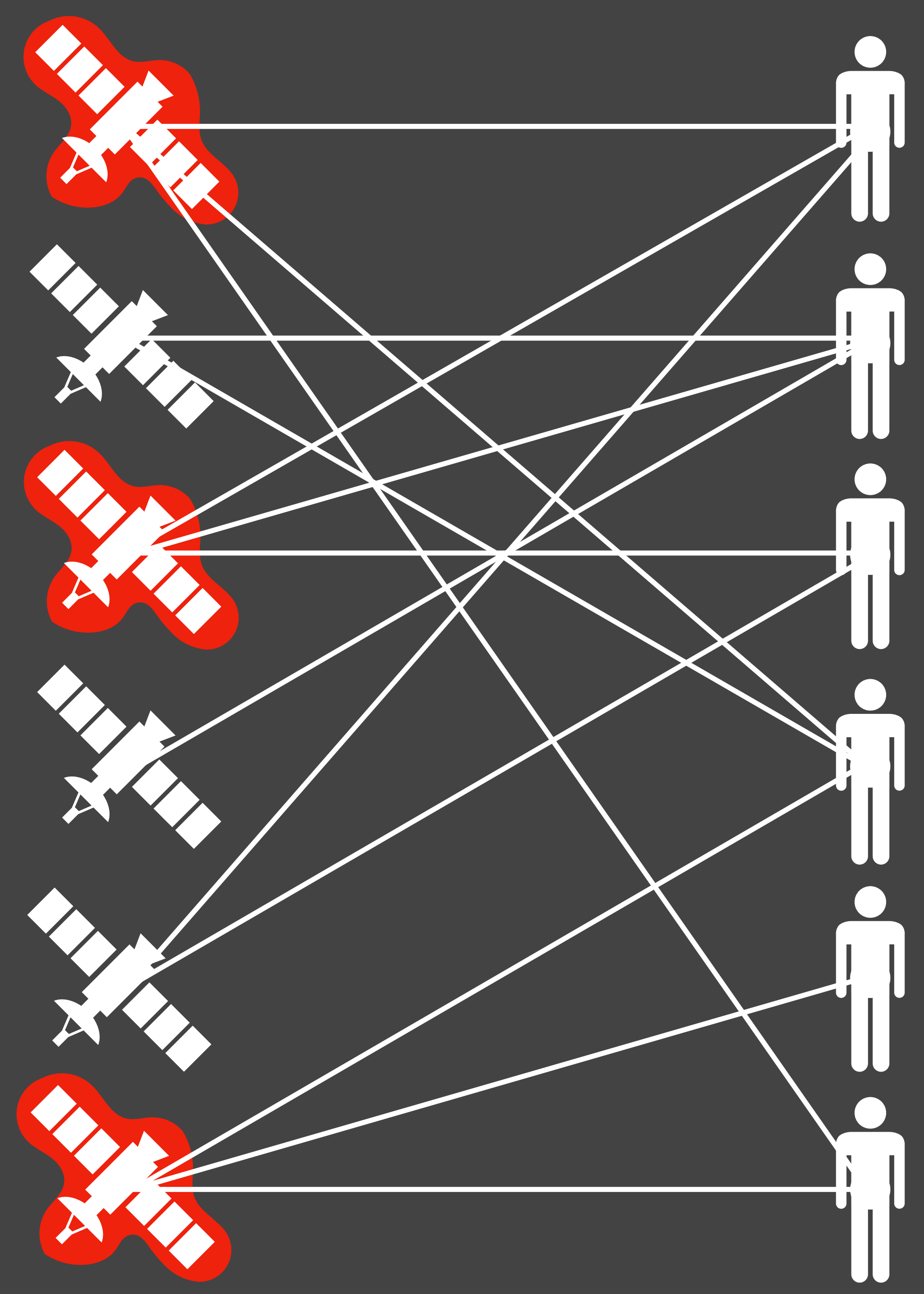


Streaming

My Work

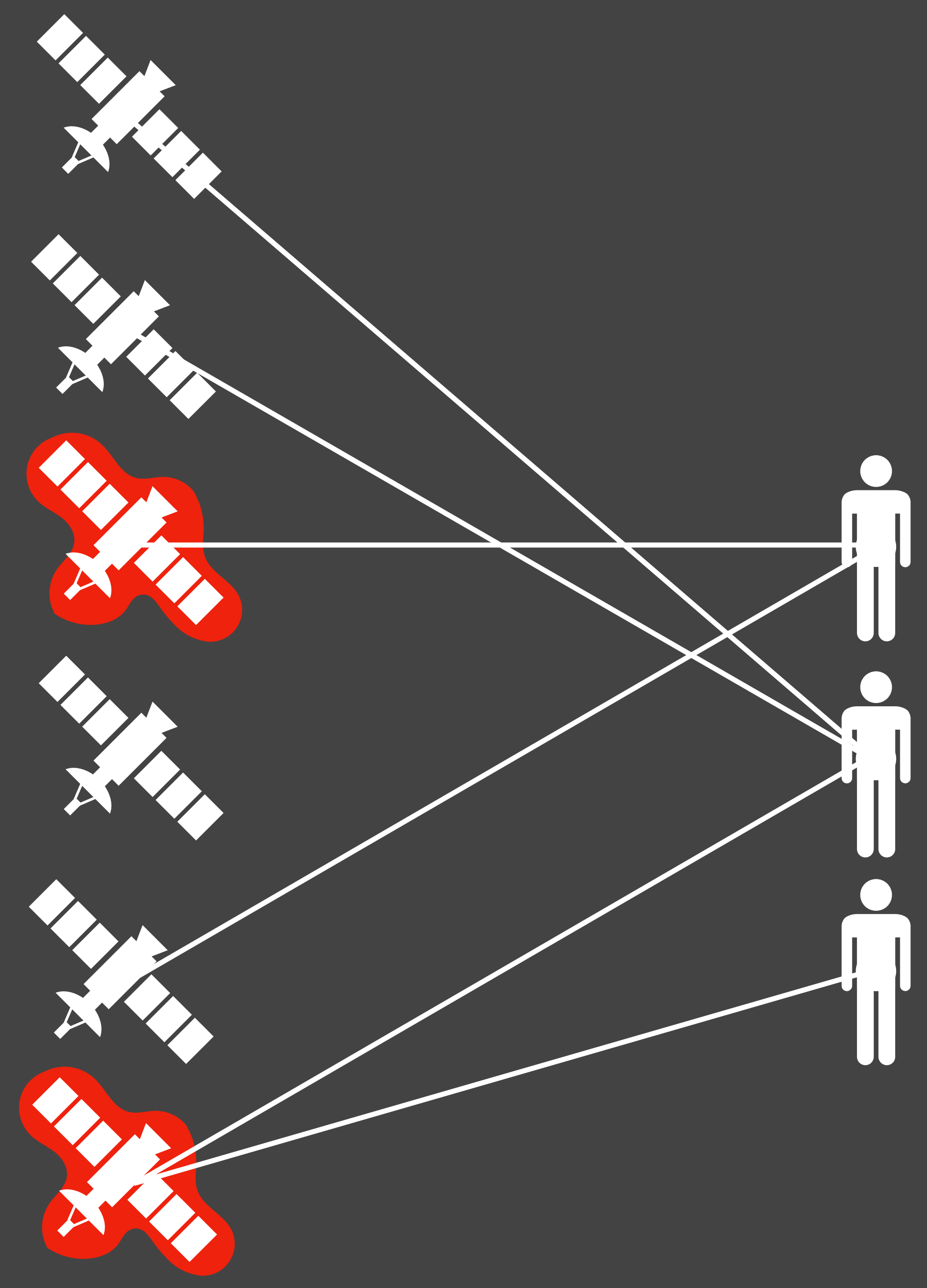
Online

No take-backs



Dynamic

Low movement

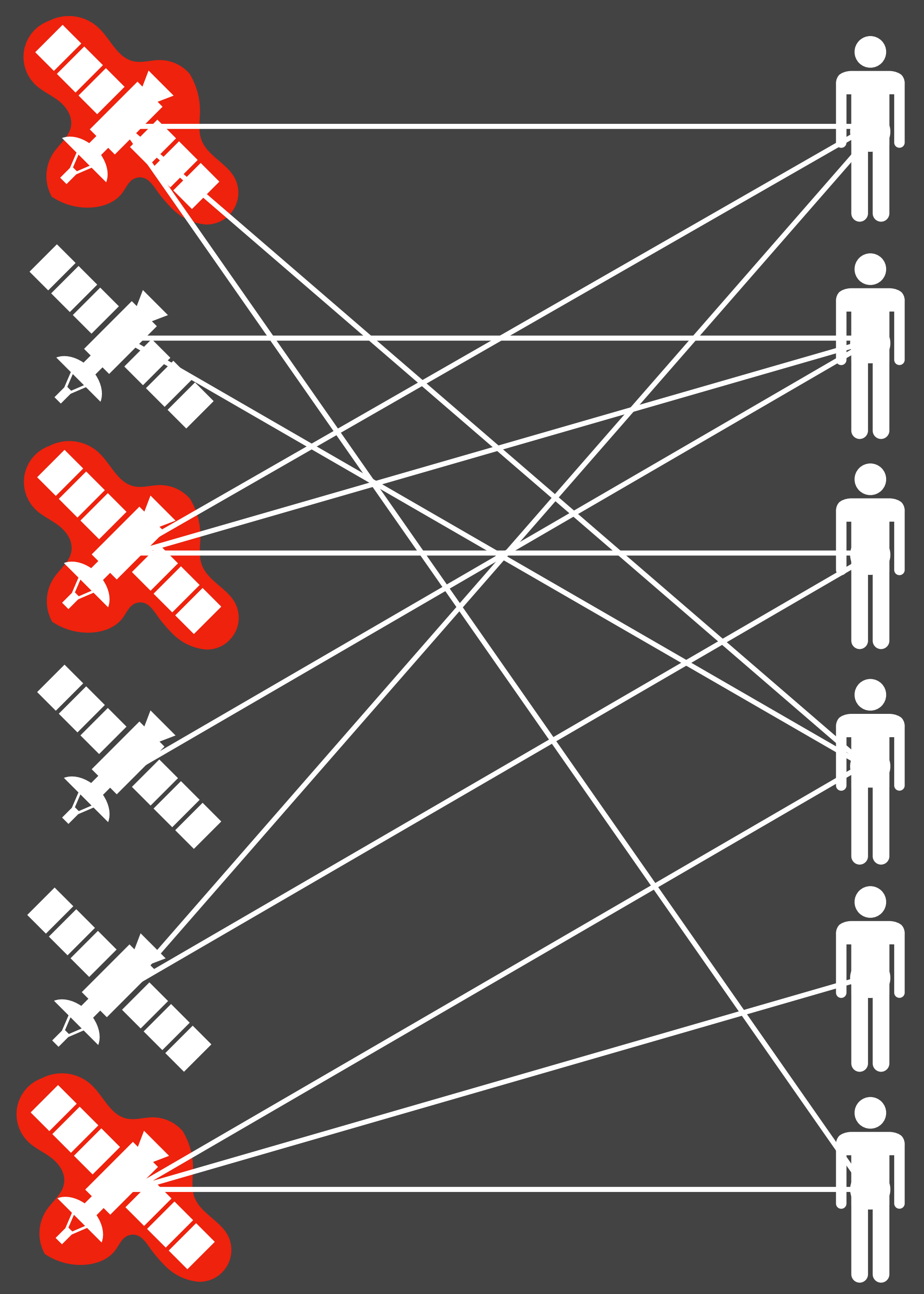


Streaming

My Work

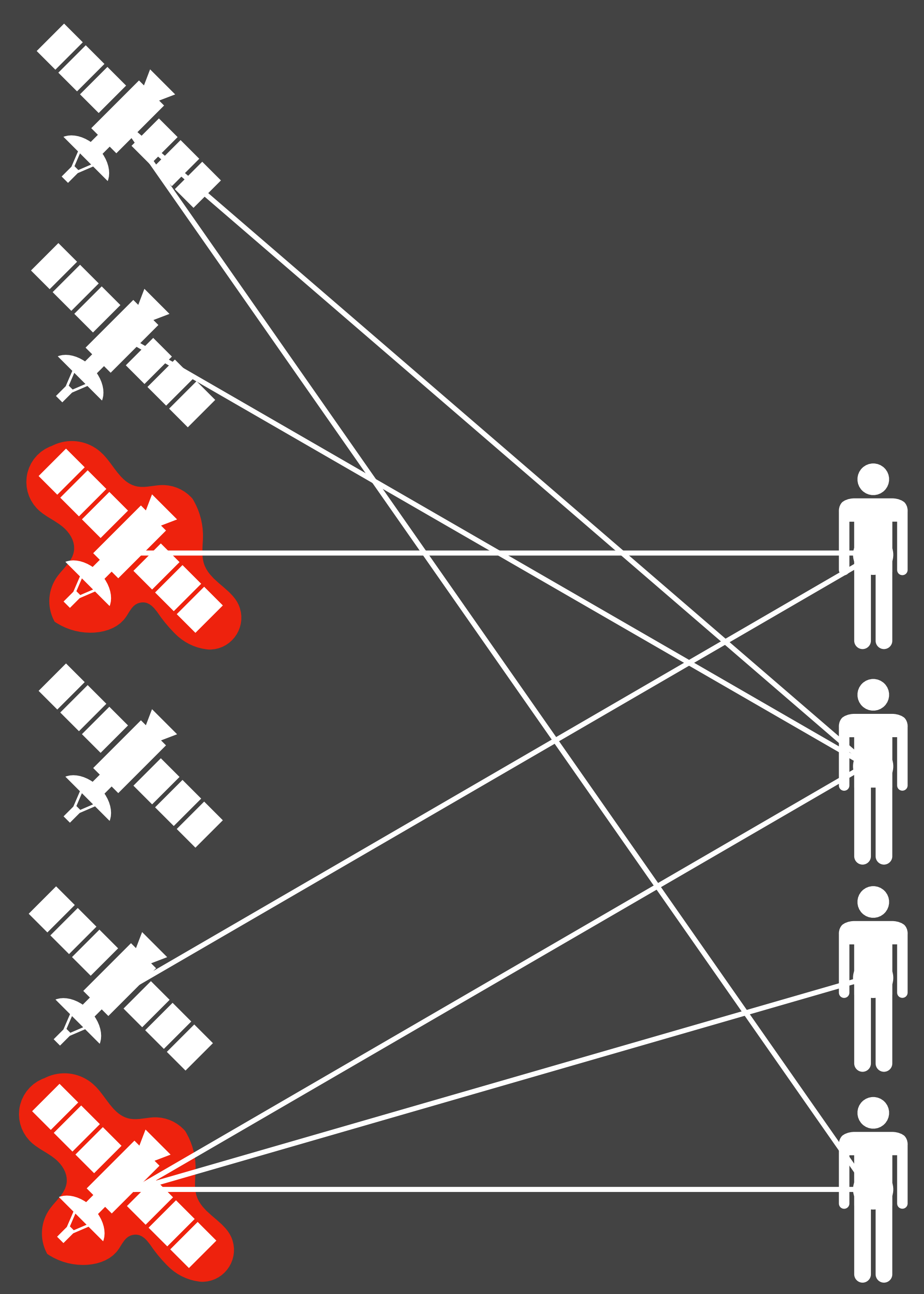
Online

No take-backs



Dynamic

Low movement

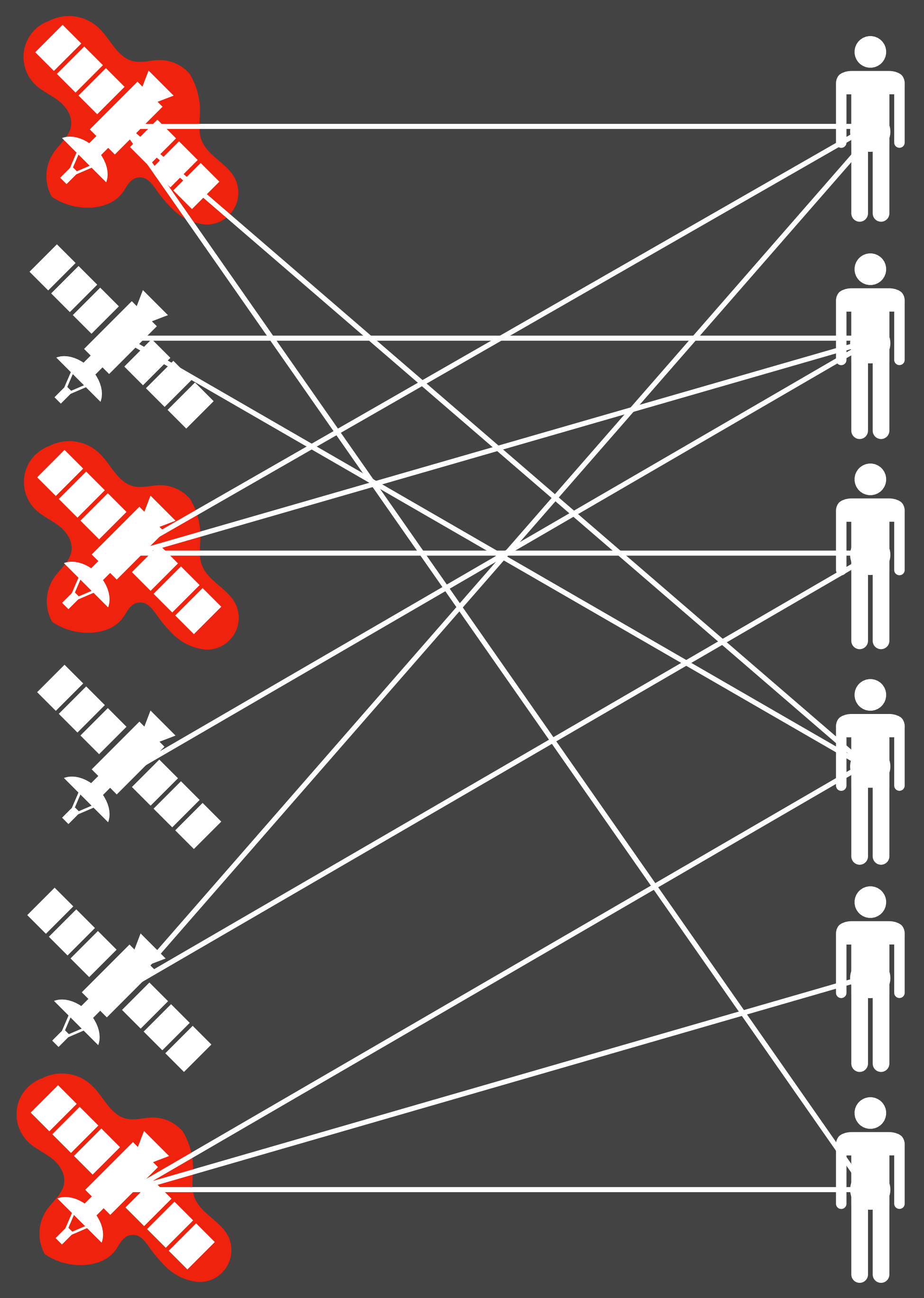


Streaming

My Work

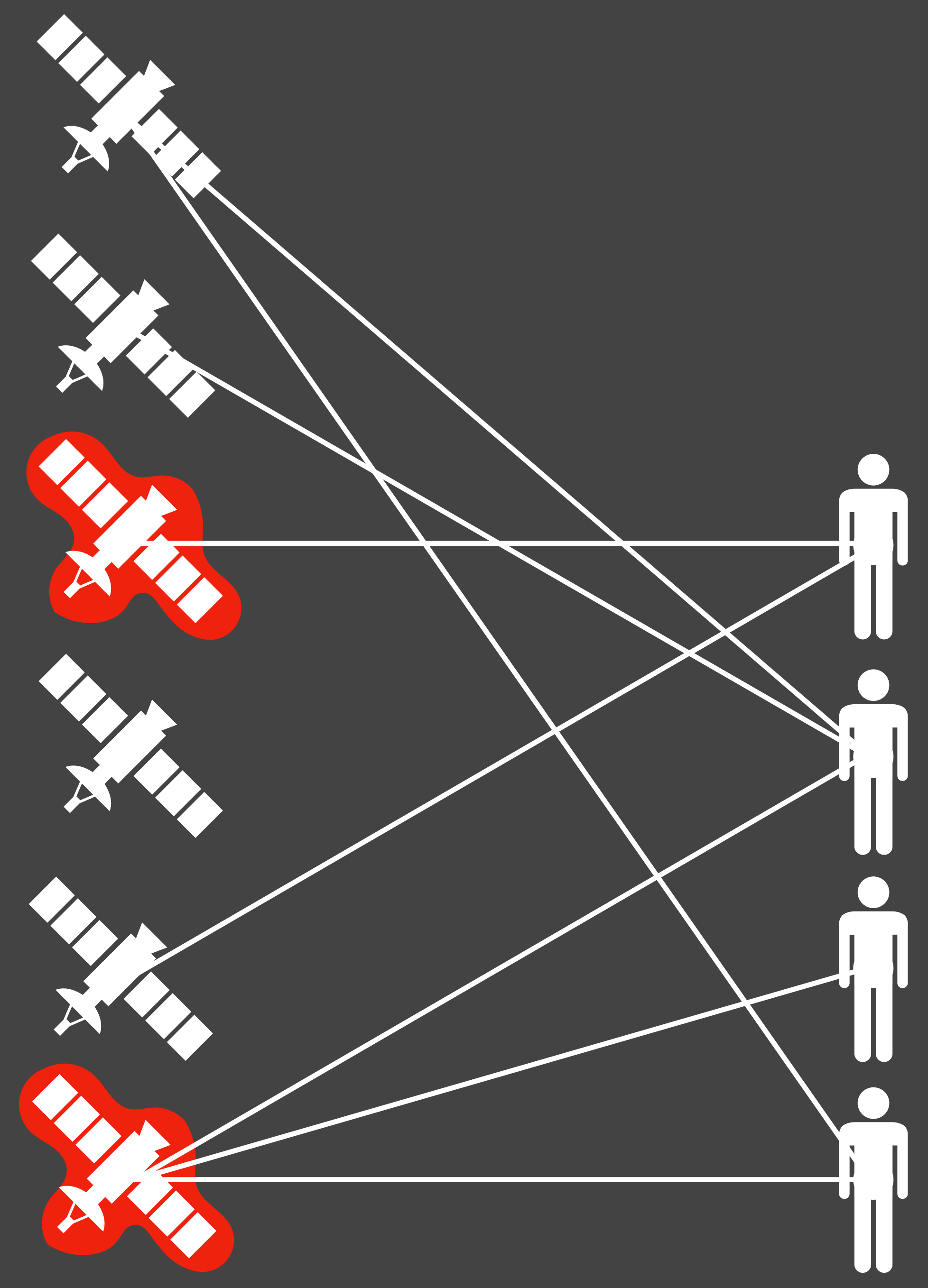
Online

No take-backs



Dynamic

Low movement



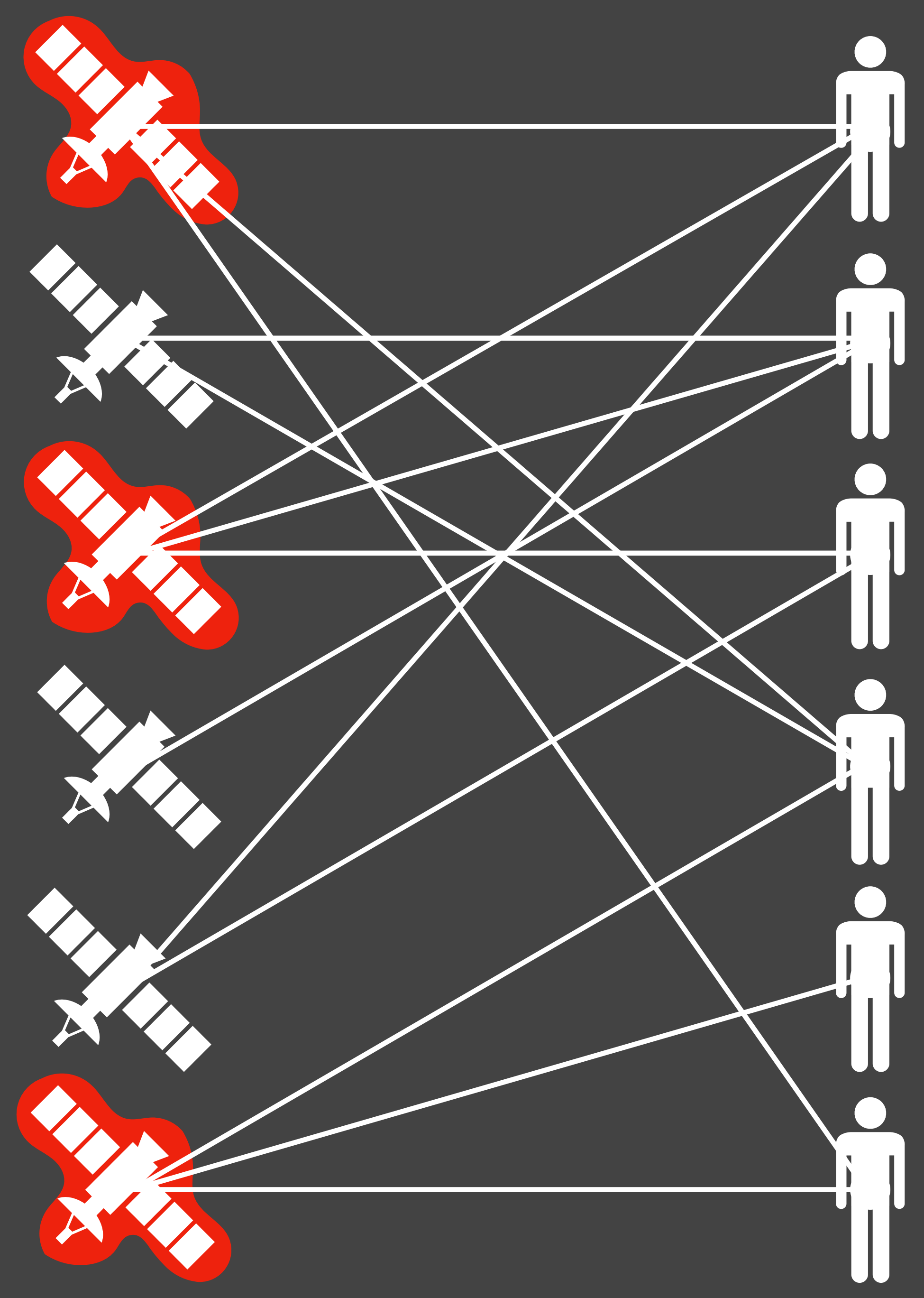
Streaming

Low memory

My Work

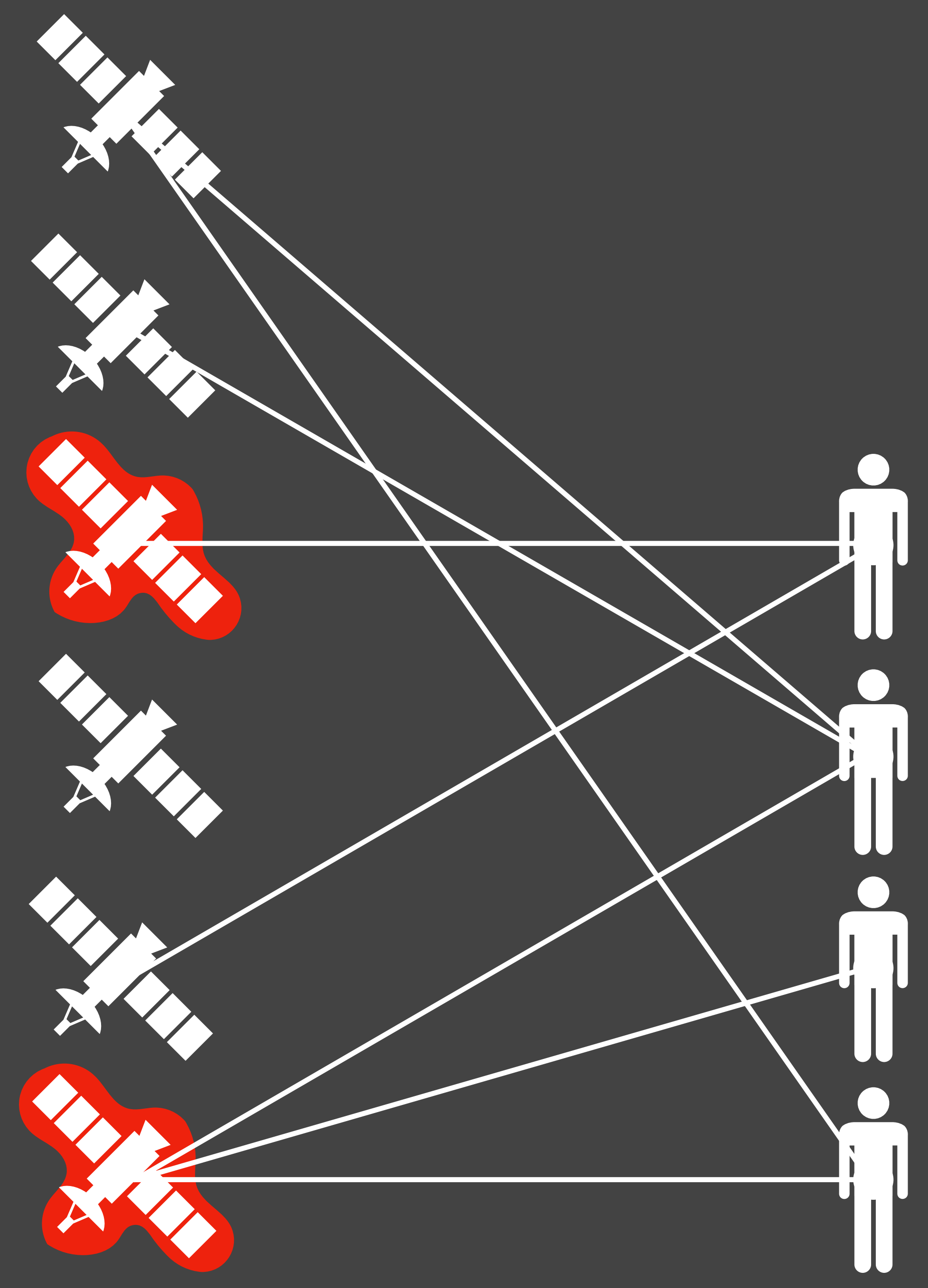
Online

No take-backs



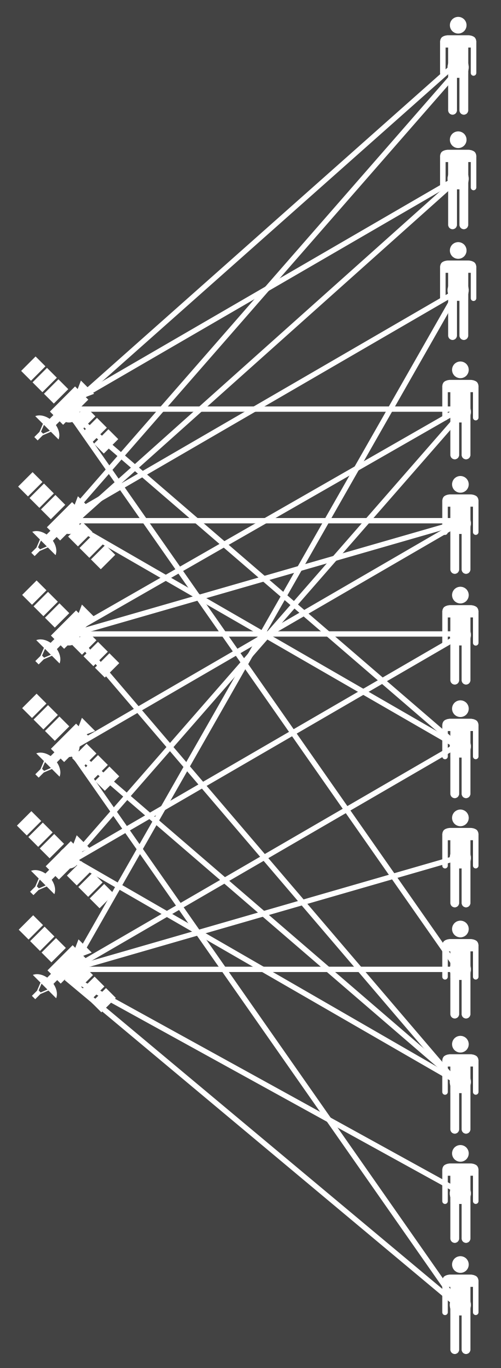
Dynamic

Low movement



Streaming

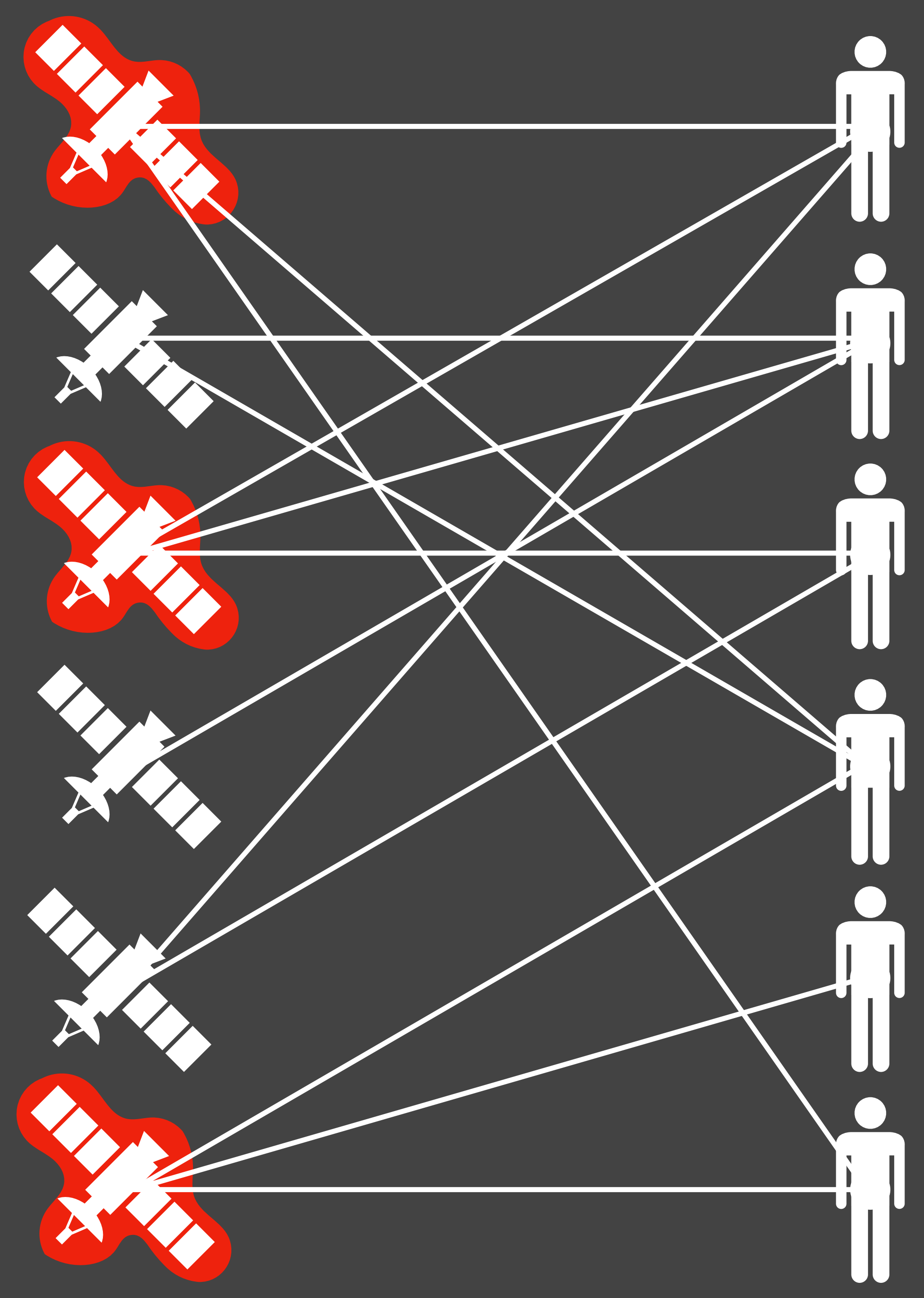
Low memory



My Work

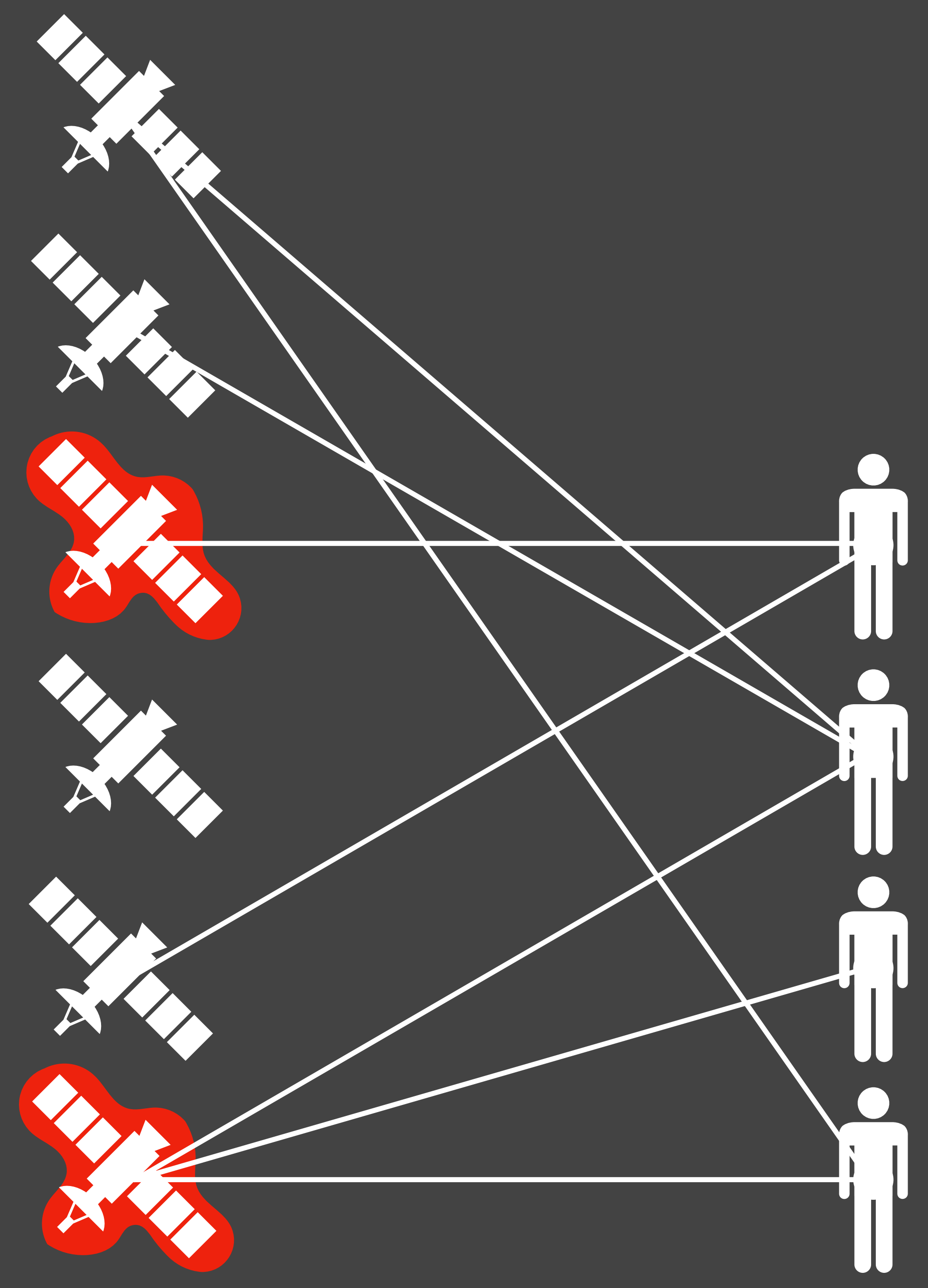
Online

No take-backs



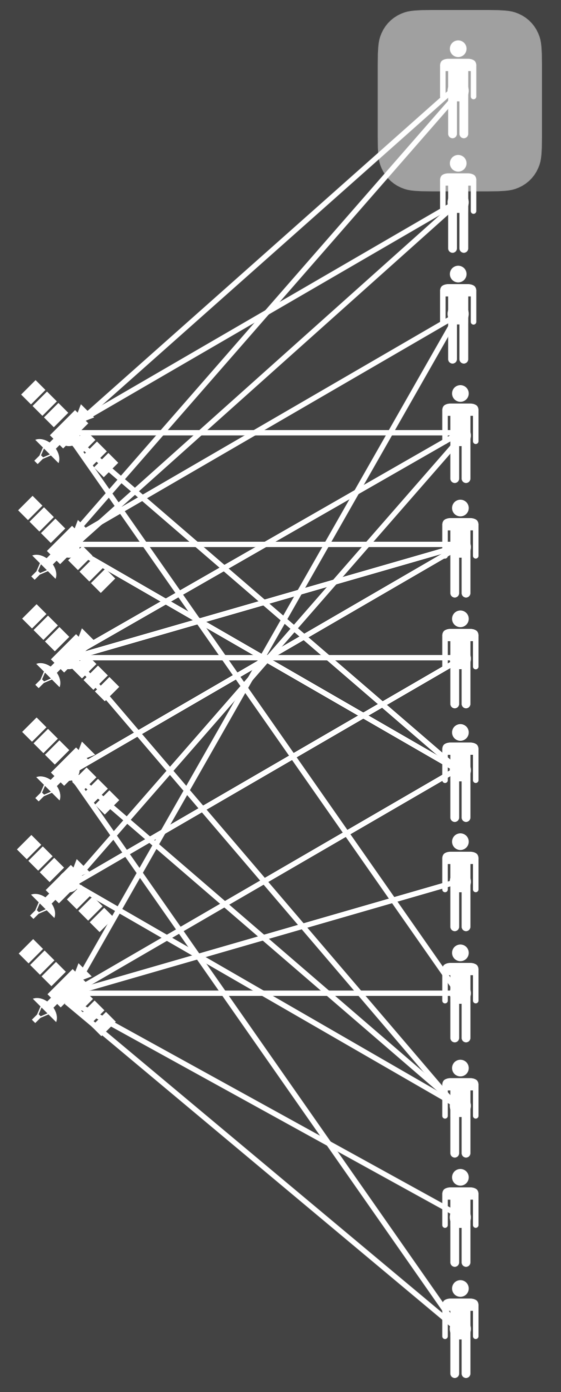
Dynamic

Low movement



Streaming

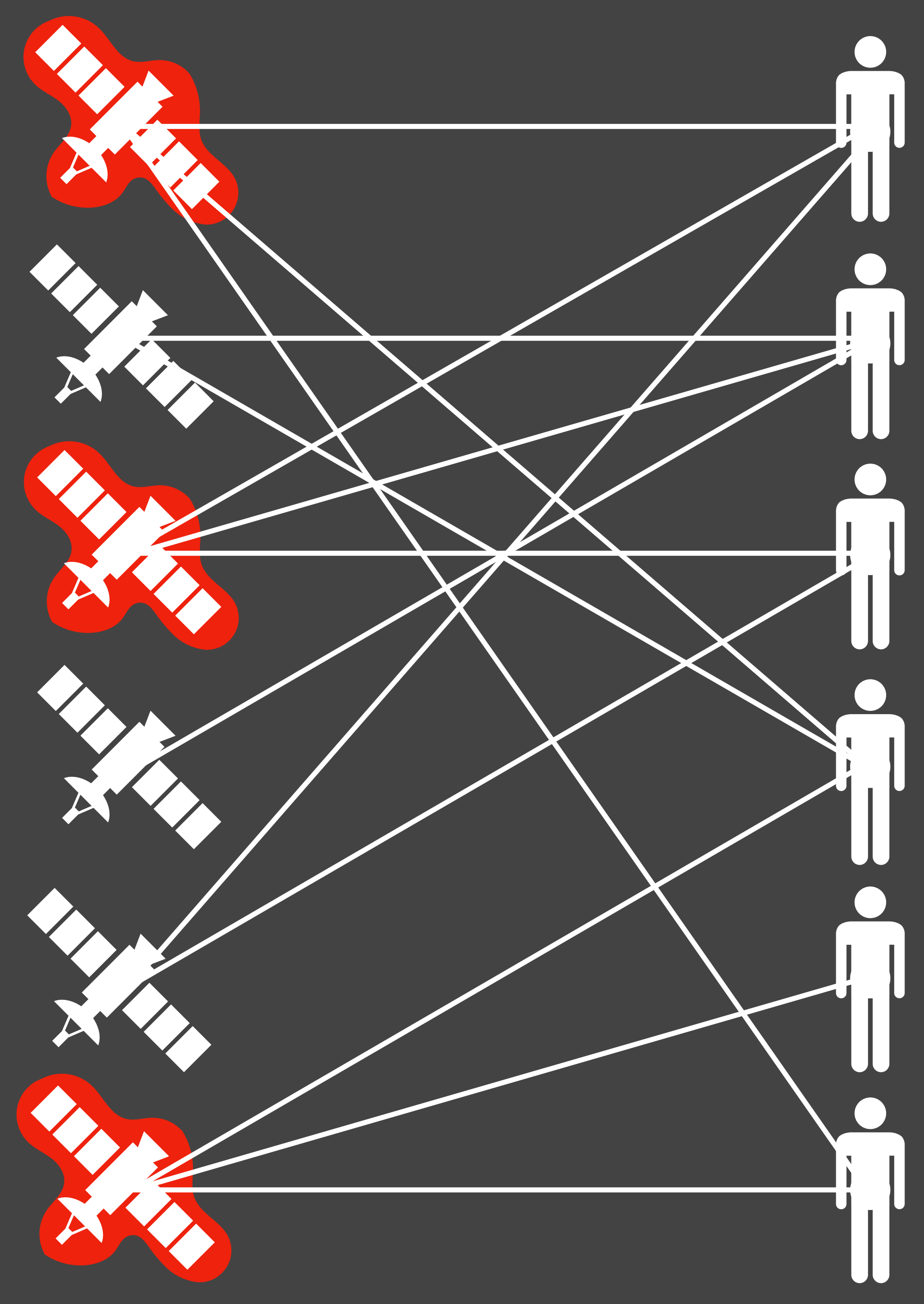
Low memory



My Work

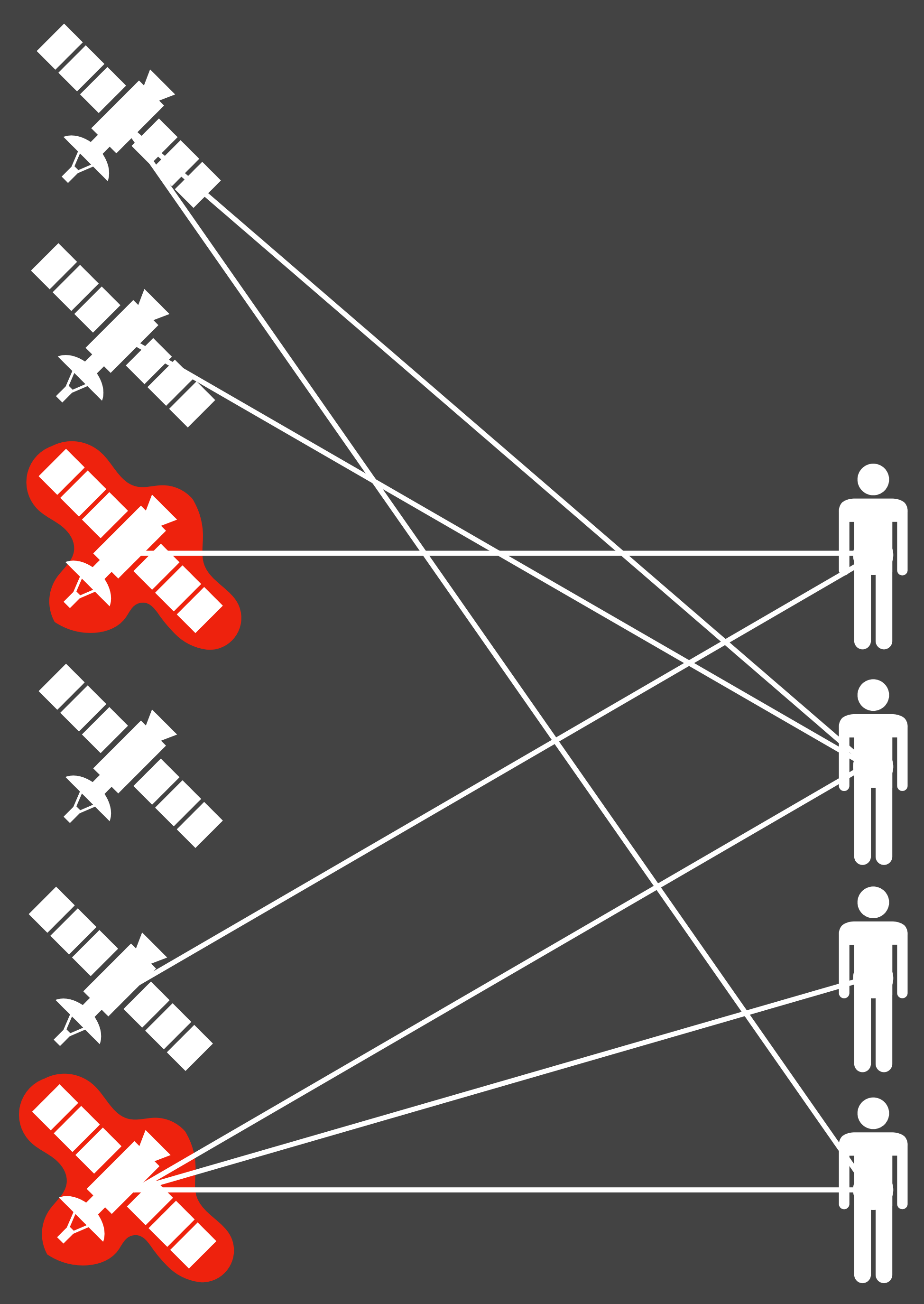
Online

No take-backs



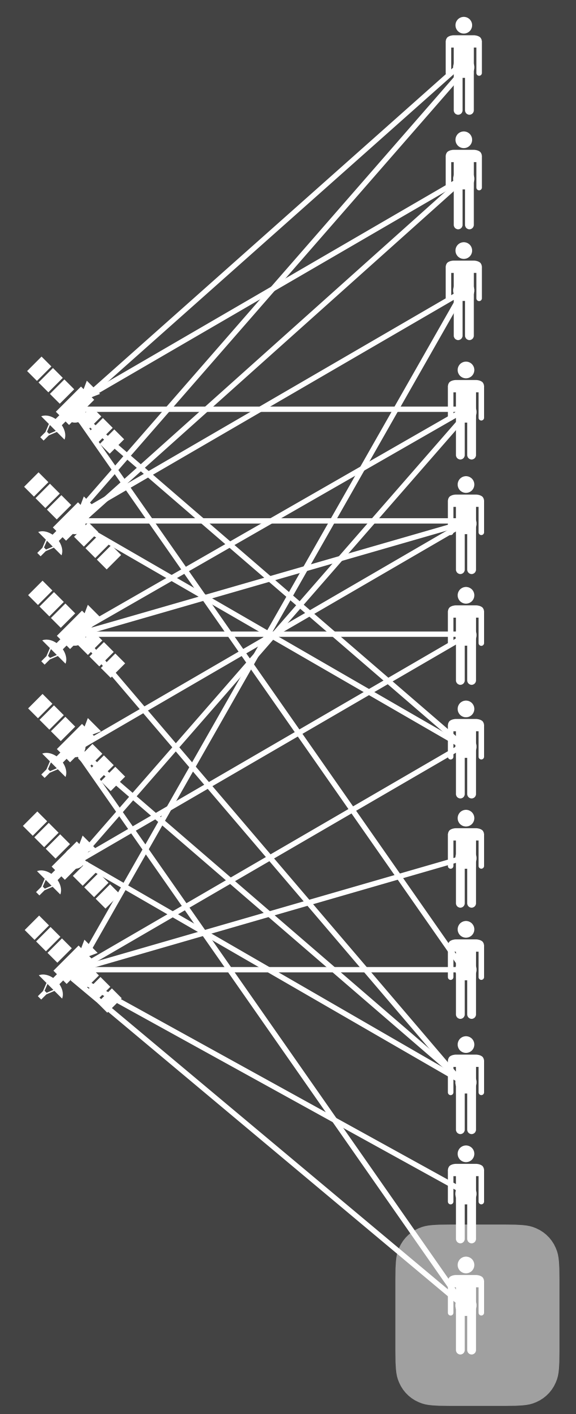
Dynamic

Low movement



Streaming

Low memory



My Work

Online

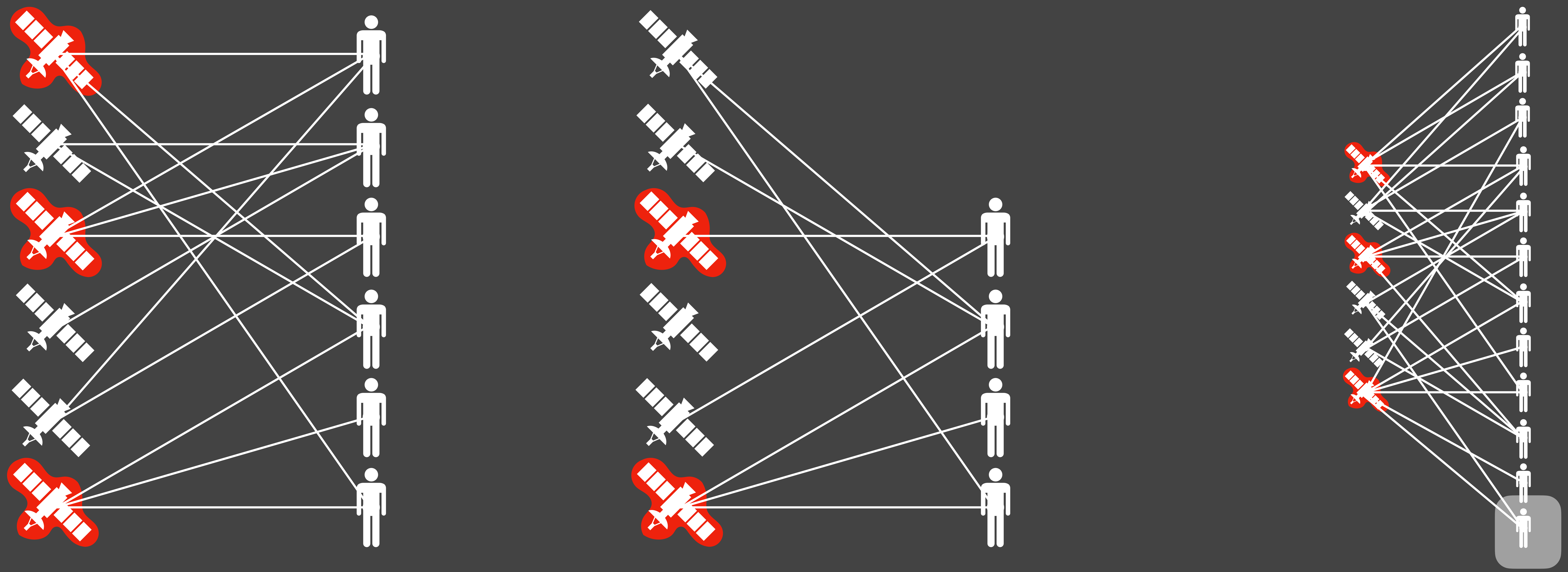
No take-backs

Dynamic

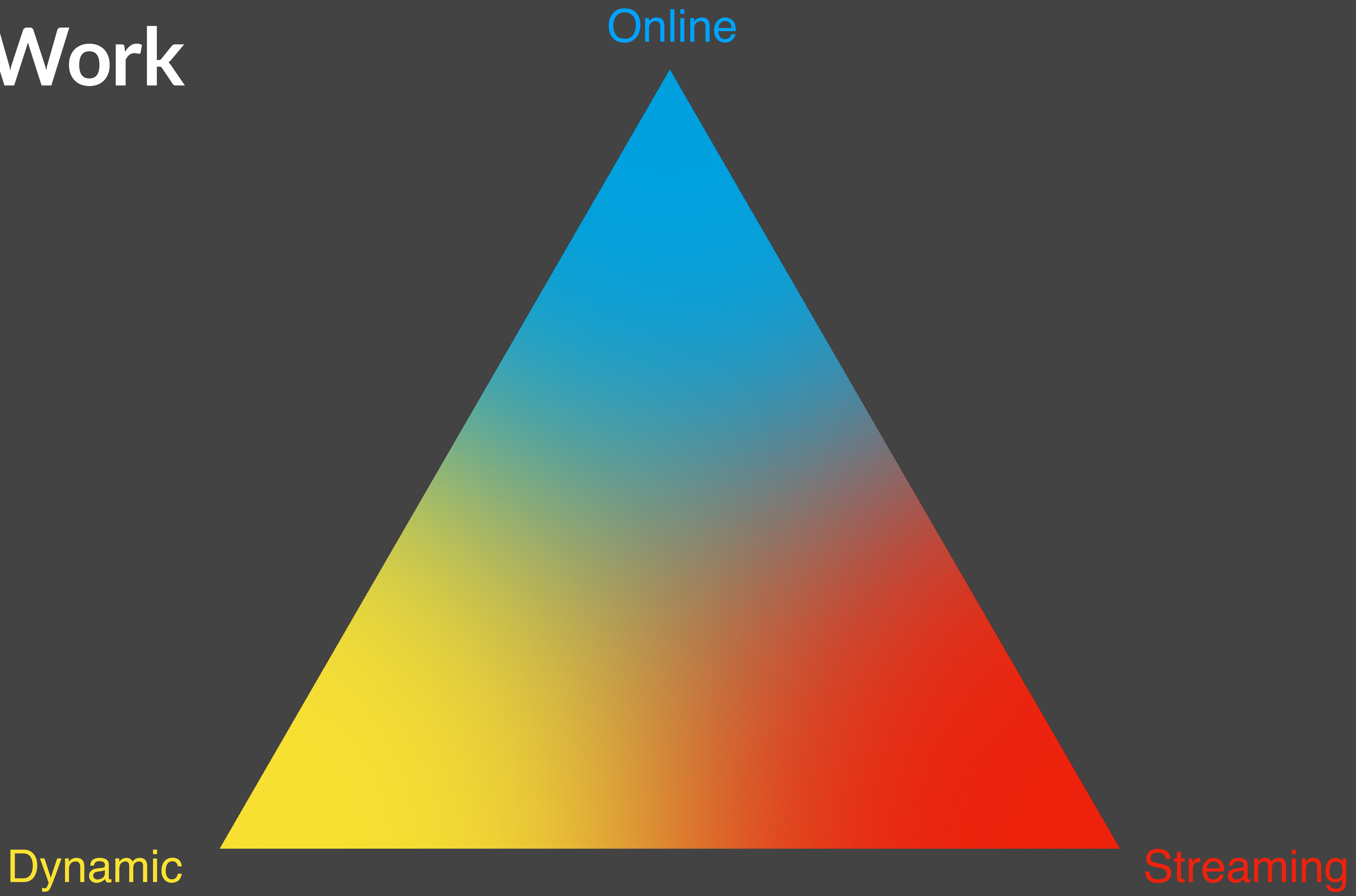
Low movement

Streaming

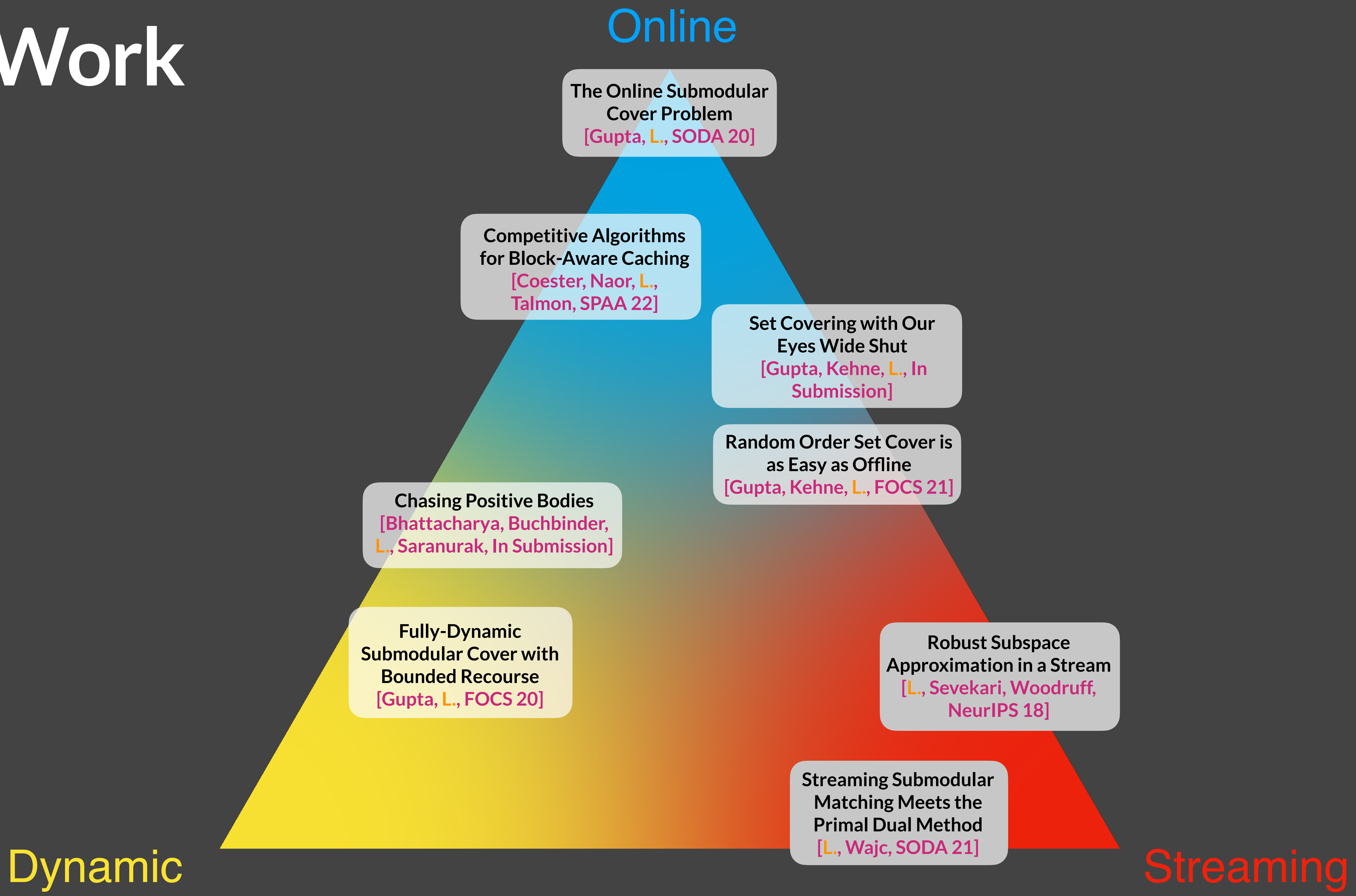
Low memory



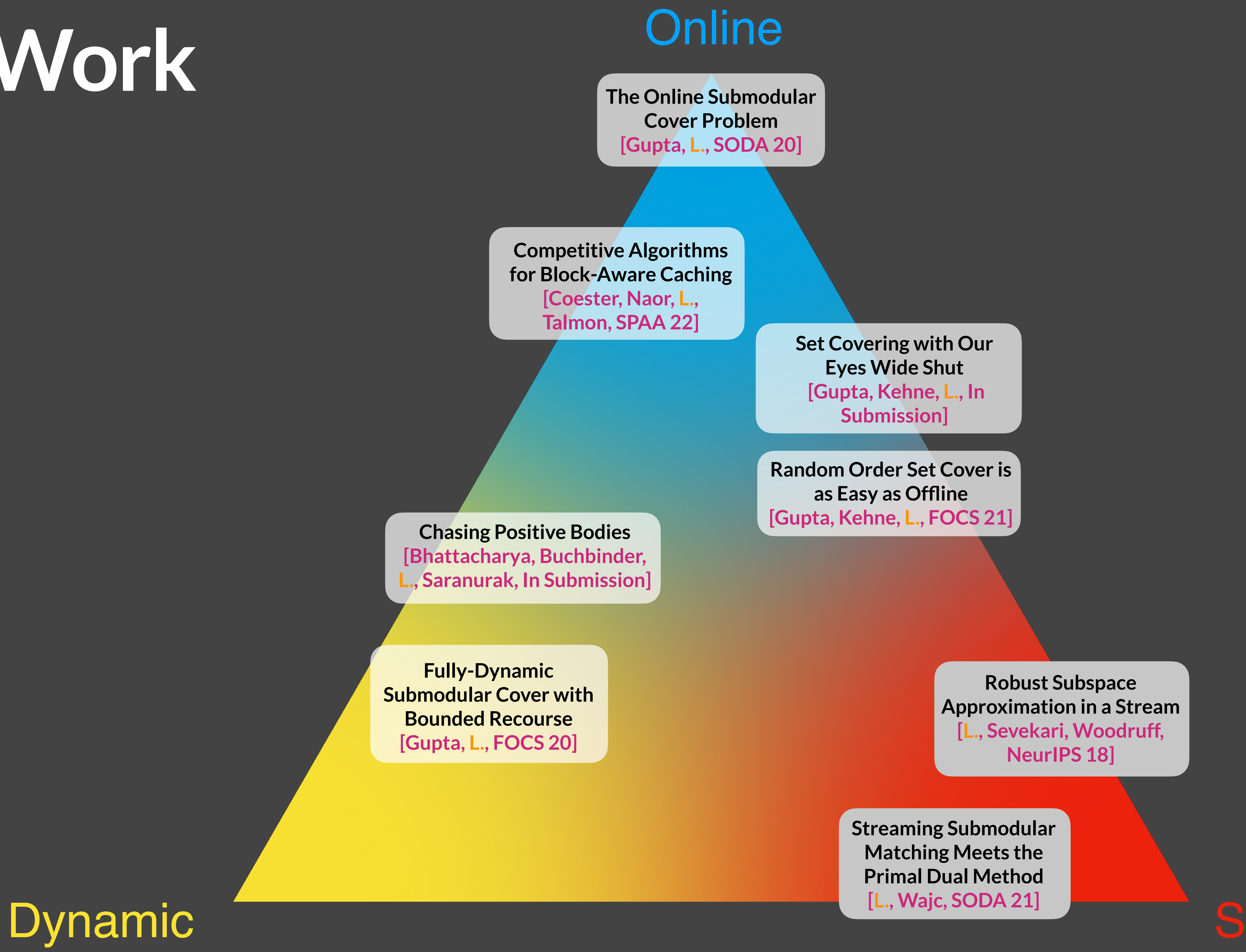
My Work



My Work


Online
The Online Submodular Cover Problem
[Gupta, L., SODA 20]
Competitive Algorithms for Block-Aware Caching
[Coester, Naor, L., Talmon, SPAA 22]
Set Covering with Our Eyes Wide Shut
[Gupta, Kehne, L., In Submission]
Random Order Set Cover is as Easy as Offline
[Gupta, Kehne, L., FOCS 21]
Chasing Positive Bodies
[Bhattacharya, Buchbinder, L., Saranurak, In Submission]
Fully-Dynamic Submodular Cover with Bounded Recourse
[Gupta, L., FOCS 20]
Robust Subspace Approximation in a Stream
[L., Sevekari, Woodruff, NeurIPS 18]
Streaming Submodular Matching Meets the Primal Dual Method
[L., Wajc, SODA 21]
Dynamic
Streaming

My Work



Finding Skewed Subcubes Under a Distribution [Gopalan, L., Wieder, ITCS 20]

FigureSeer: Parsing Result-Figures in Research Papers [Siegel, Horvitz, L., Divvala, Farhadi, ECCV 16]

Beyond Sentential Semantic Parsing: Tackling the Math SAT with a Cascade of Tree Transducers [Hopkins, Petrscu-Prahova, L., Le Bras, Herrasti, Joshi, EMNLP 17]

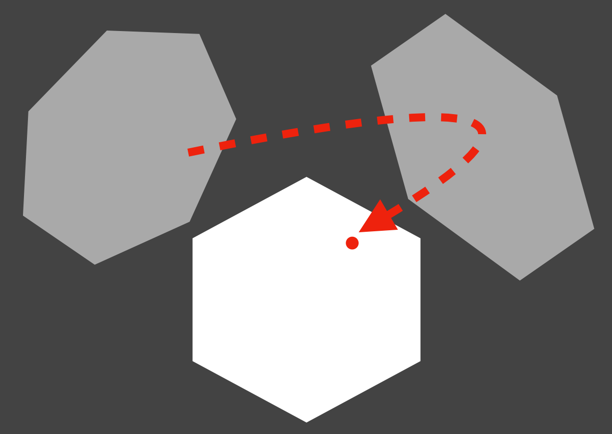
… and others in AI, ML, Fairness

Outline

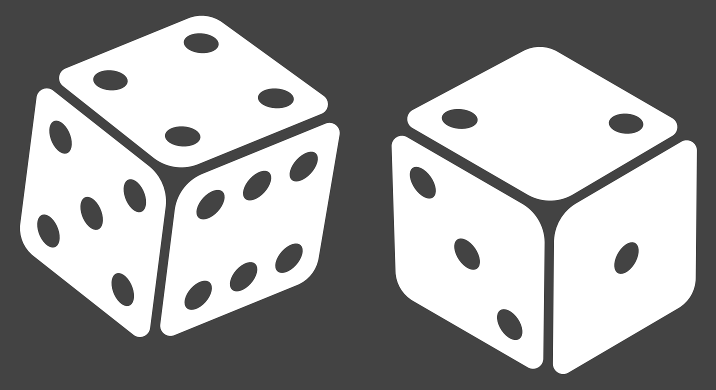
Theme I — Submodular Optimization

f(🍕 | 🥕) ≥ f(🍕 | 🥕, 🍩)

Theme II — Stable Algorithms



Theme III — Beyond Worst-Case Analysis



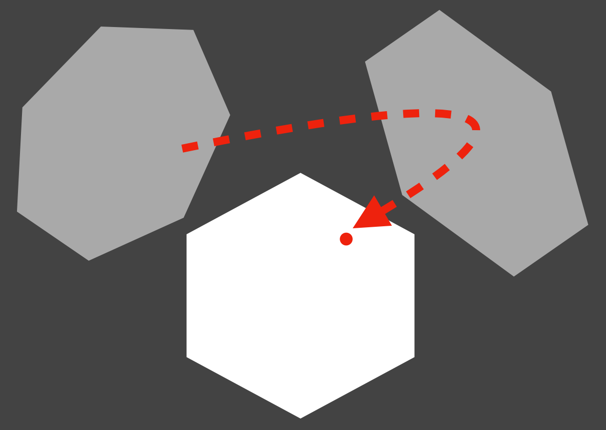
Conclusion

Outline

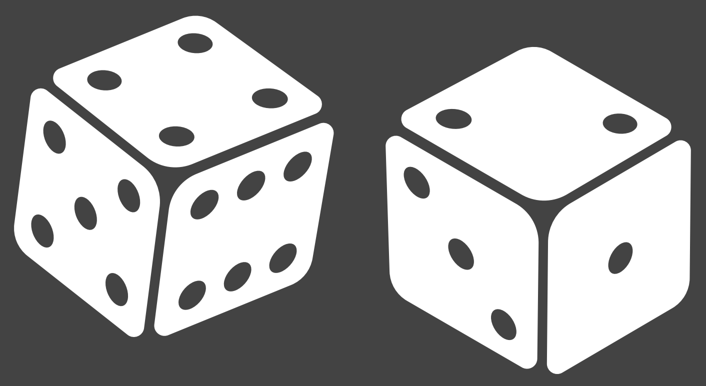
Theme I — Submodular Optimization

f(🍕 | 🥕) ≥ *f*(🍕 | 🥕, 🍩)

Theme II — Stable Algorithms



Theme III — Beyond Worst-Case Analysis



Conclusion

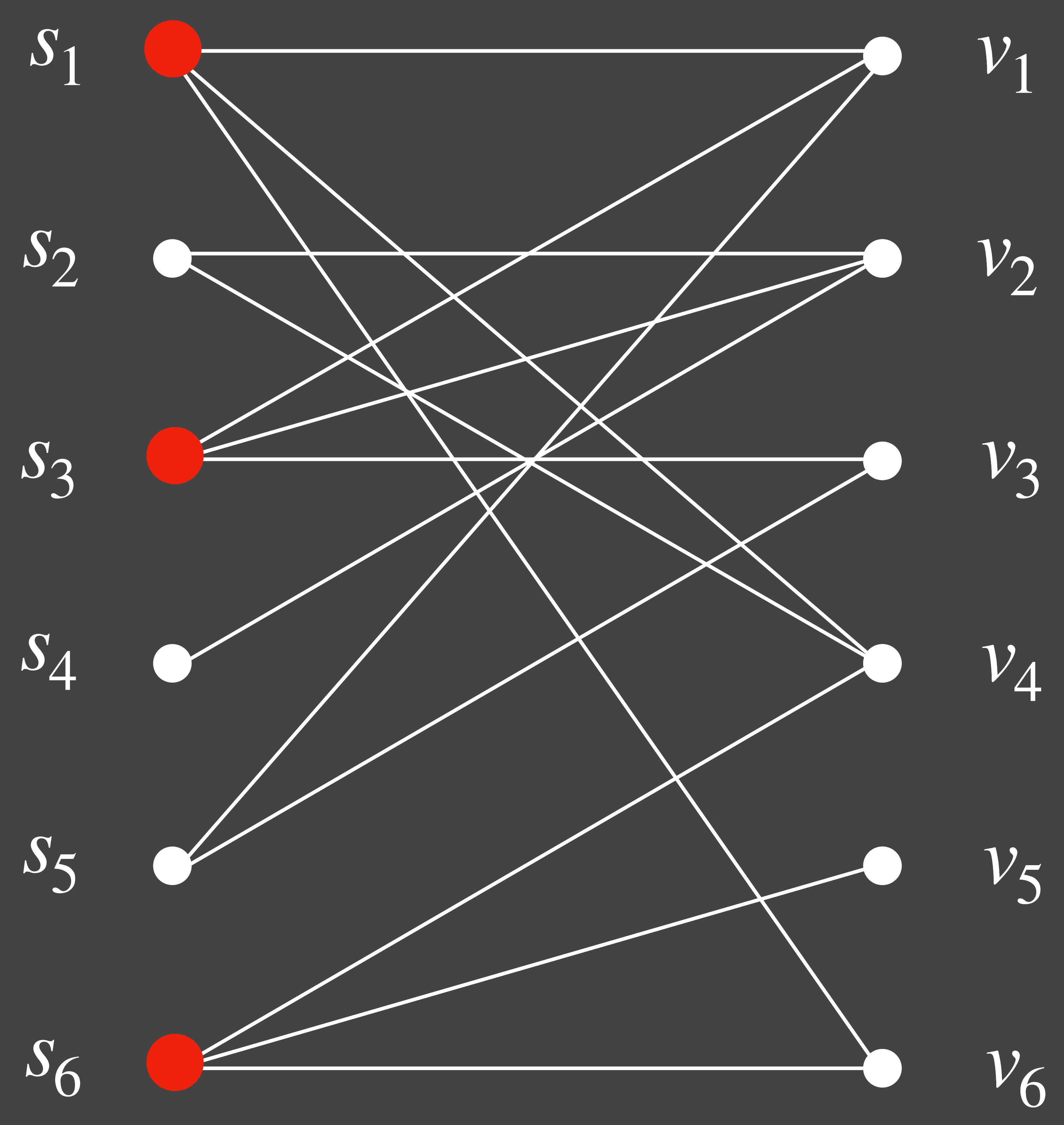
Theme I — Submodular Optimization

Beyond Set Cover

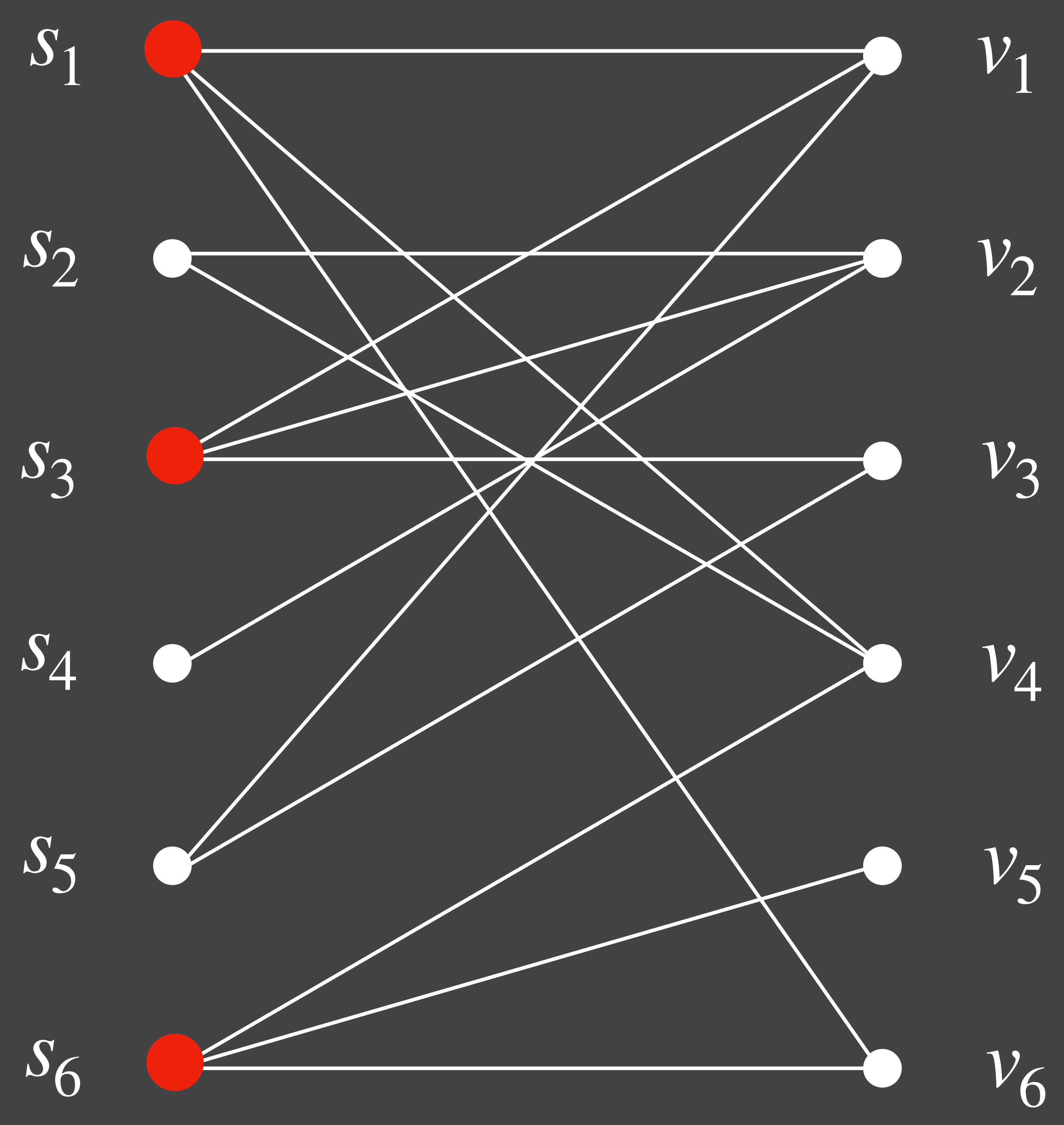
Q: What general classes of optimization problems can we solve online?

Beyond Set Cover

Q: What general classes of optimization problems can we solve online?

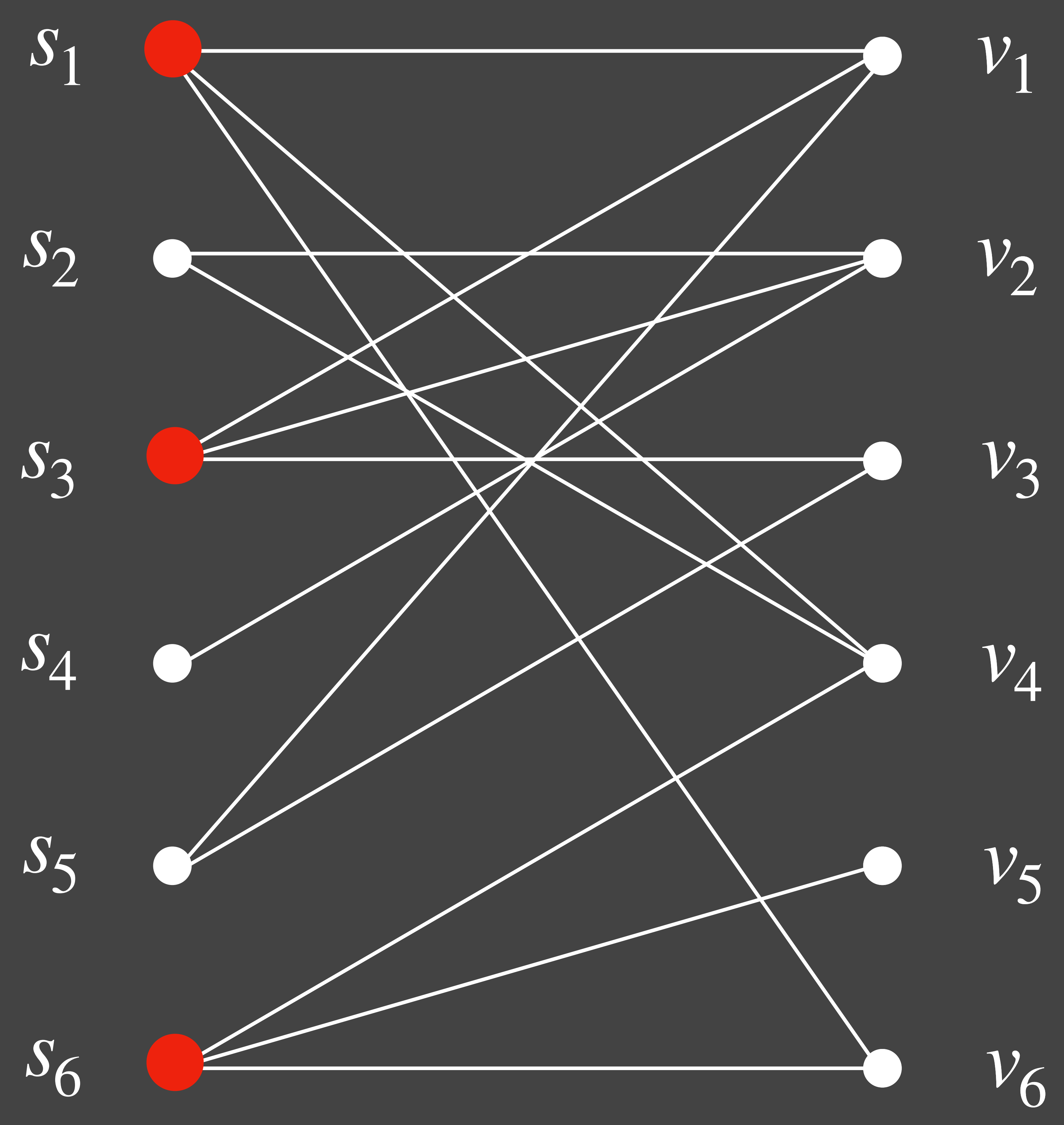


Abstracting the Problem



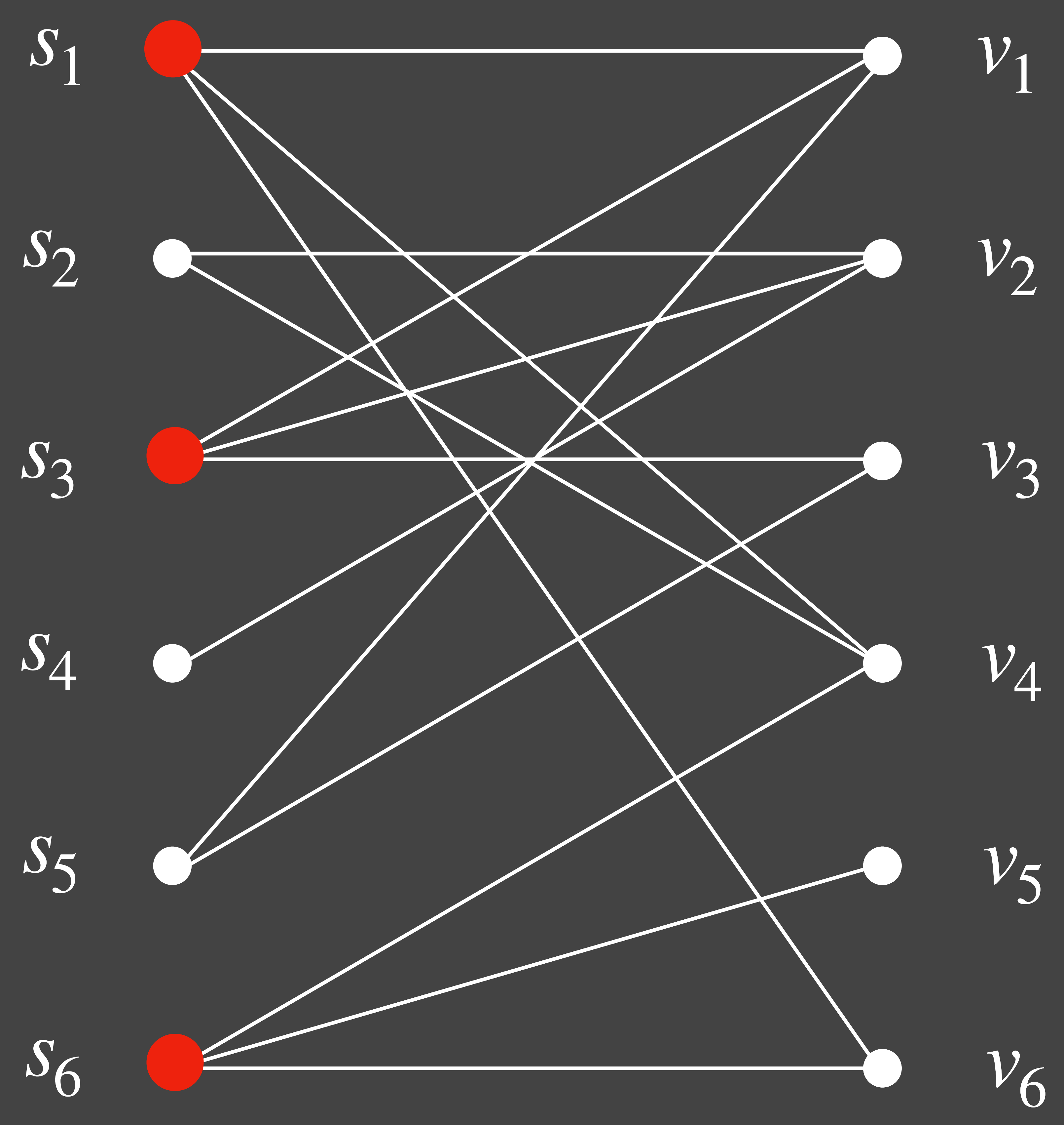
Abstracting the Problem

- Universe of choices: $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$



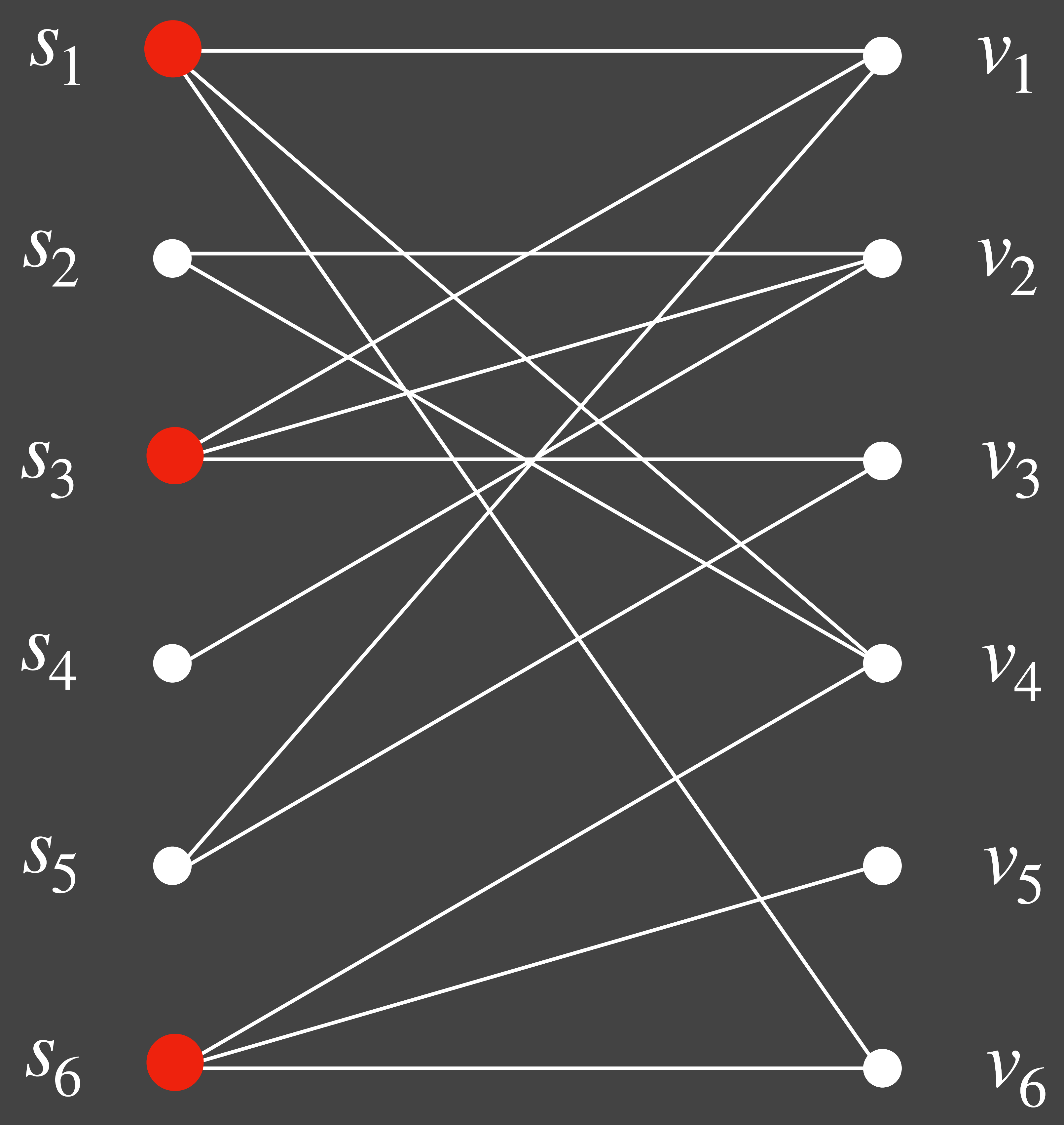
Abstracting the Problem

- Universe of choices: $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$
- Solution: $S \subseteq \mathcal{S}$



Abstracting the Problem

- Universe of choices: $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$
- Solution: $S \subseteq \mathcal{S}$
- Cost: $c(S)$



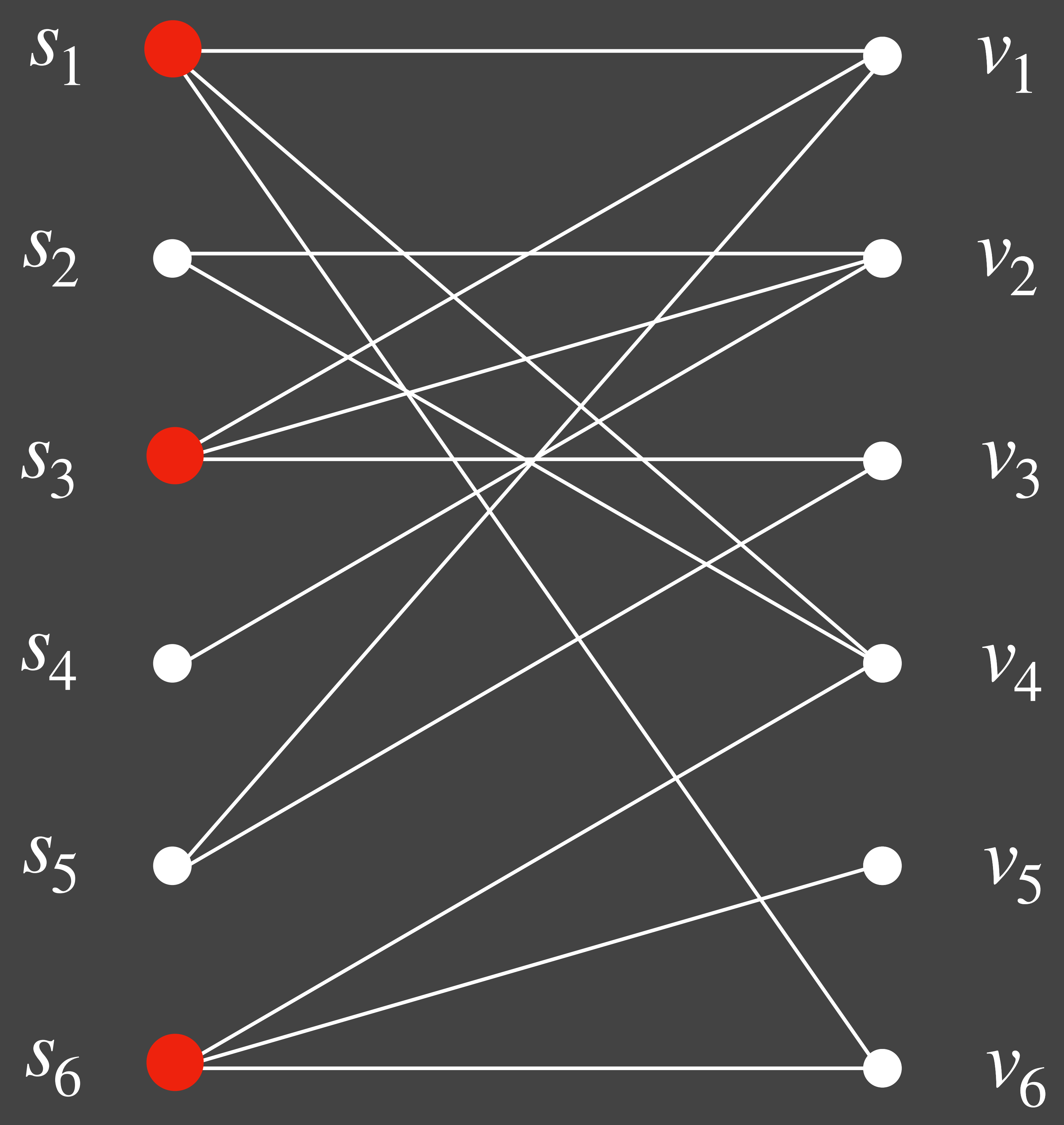
Abstracting the Problem

- Universe of choices: $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$

- Solution: $S \subseteq \mathcal{S}$

- Cost: $c(S)$

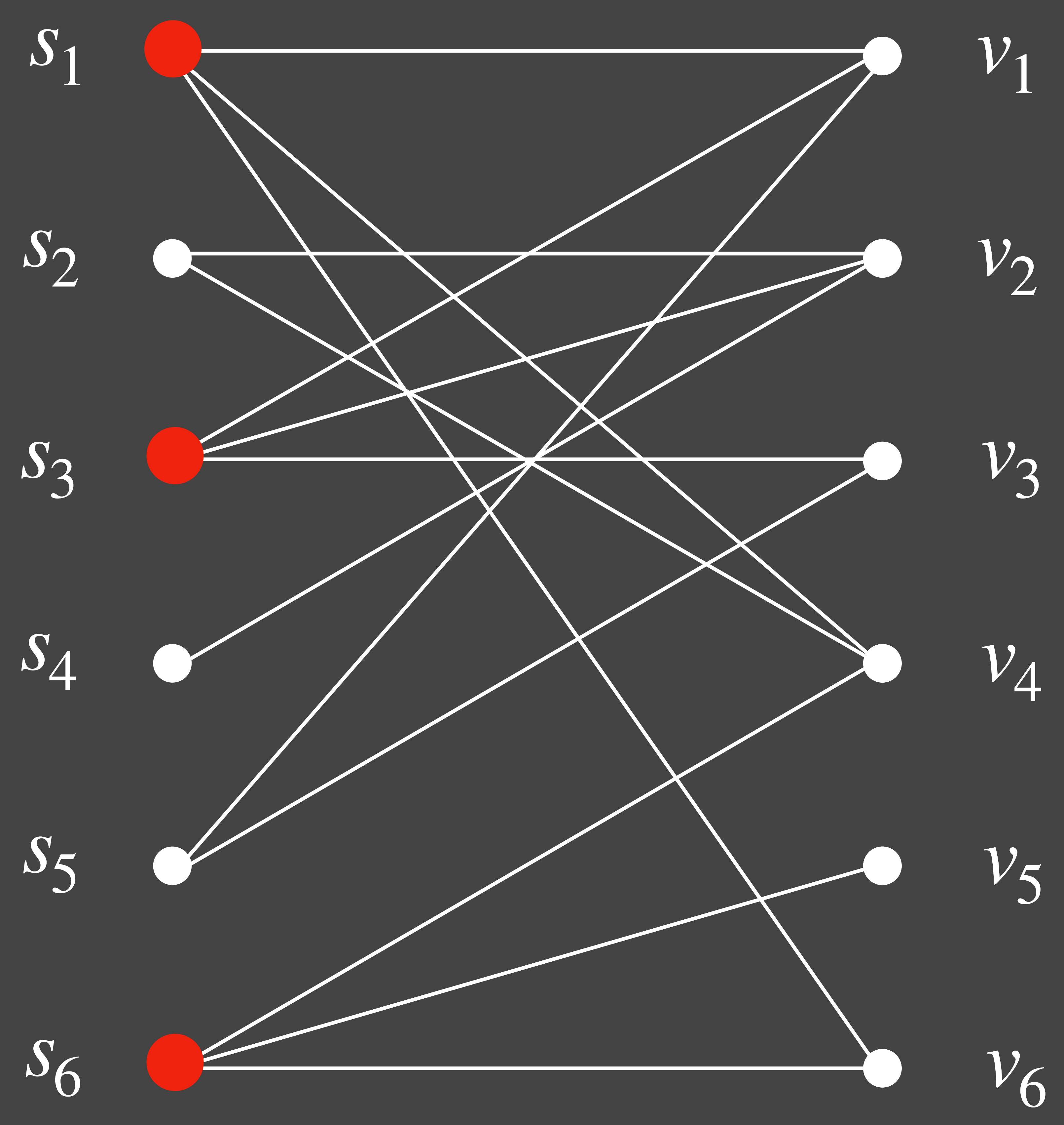
- Coverage "Quality": $f(S)$



Abstracting the Problem

- Universe of choices: $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$
- Solution: $S \subseteq \mathcal{S}$
- Cost: $c(S)$
- Coverage "Quality": $f(S)$

Want **min cost** solution with **max coverage**!



Abstracting the Problem

- Universe of choices: $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$
- Solution: $S \subseteq \mathcal{S}$
- Cost: $c(S)$
- Coverage "Quality": $f(S)$

$$\min_{S \subseteq \mathcal{S}} c(S)$$

$$f(S) \geq n$$

Want **min cost** solution with **max coverage**!

Abstracting the Problem

- Universe of choices: $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$
- Solution: $S \subseteq \mathcal{S}$
- Cost: $c(S)$
- Coverage "Quality": $f(S)$

$$\boxed{\min_{S \subseteq \mathcal{S}} c(S) \qquad f(S) \geq n}$$

Want **min cost** solution with **max coverage**!

$f : 2^{\mathcal{N}} \to \mathbb{R}$ is **monotone**, **nonnegative** and **<u>submodular</u>**.

Abstracting the Problem

a.k.a. Submodular Cover [Wolsey 82]

- Universe of choices: $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$
- Solution: $S \subseteq \mathcal{S}$
- Cost: $c(S)$
- Coverage "Quality": $f(S)$

$$\min_{S \subseteq \mathcal{S}} c(S)$$

$$f(S) \geq n$$

Want **min cost** solution with **max coverage**!

$f : 2^{\mathcal{N}} \to \mathbb{R}$ is **monotone**, **nonnegative** and **submodular**.

Abstracting the Problem

a.k.a. Submodular Cover [Wolsey 82]

- Universe of choices: $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$
- Solution: $S \subseteq \mathcal{S}$
- Cost: $c(S)$
- Coverage "Quality": $f(S)$

$$\min_{S \subseteq \mathcal{S}} c(S)$$

$$f(S) \geq n$$

Want **min cost** solution with **max coverage**!

We will port this online!

$f : 2^{\mathcal{N}} \to \mathbb{R}$ is **monotone**, **nonnegative** and **submodular**.

Submodularity

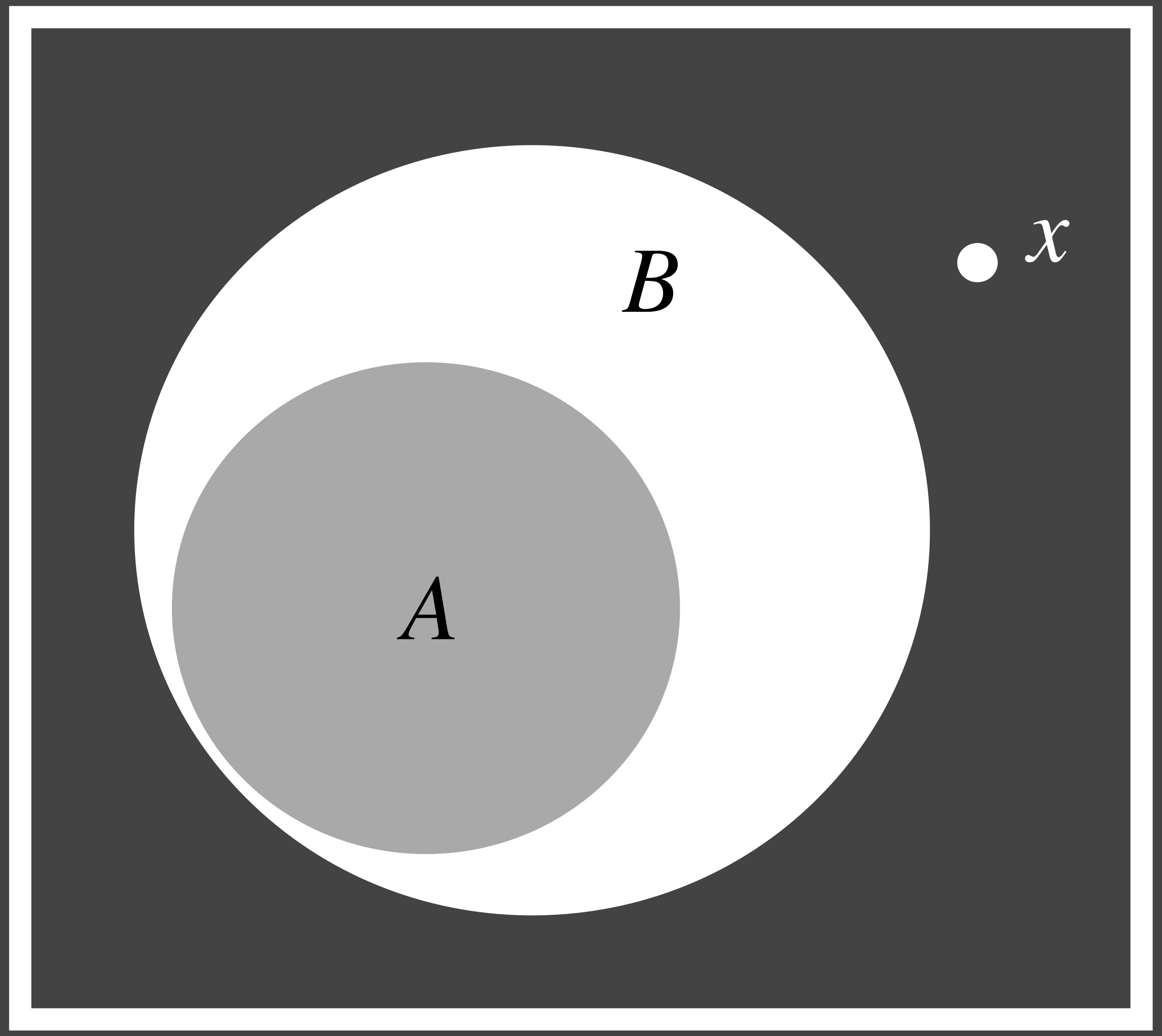
Submodularity

a.k.a. “Decreasing Marginal Returns!”

Submodularity

a.k.a. "Decreasing Marginal Returns!"

Definition: f is ***submodular*** *if*, $\forall A \subseteq B, x \notin B,$

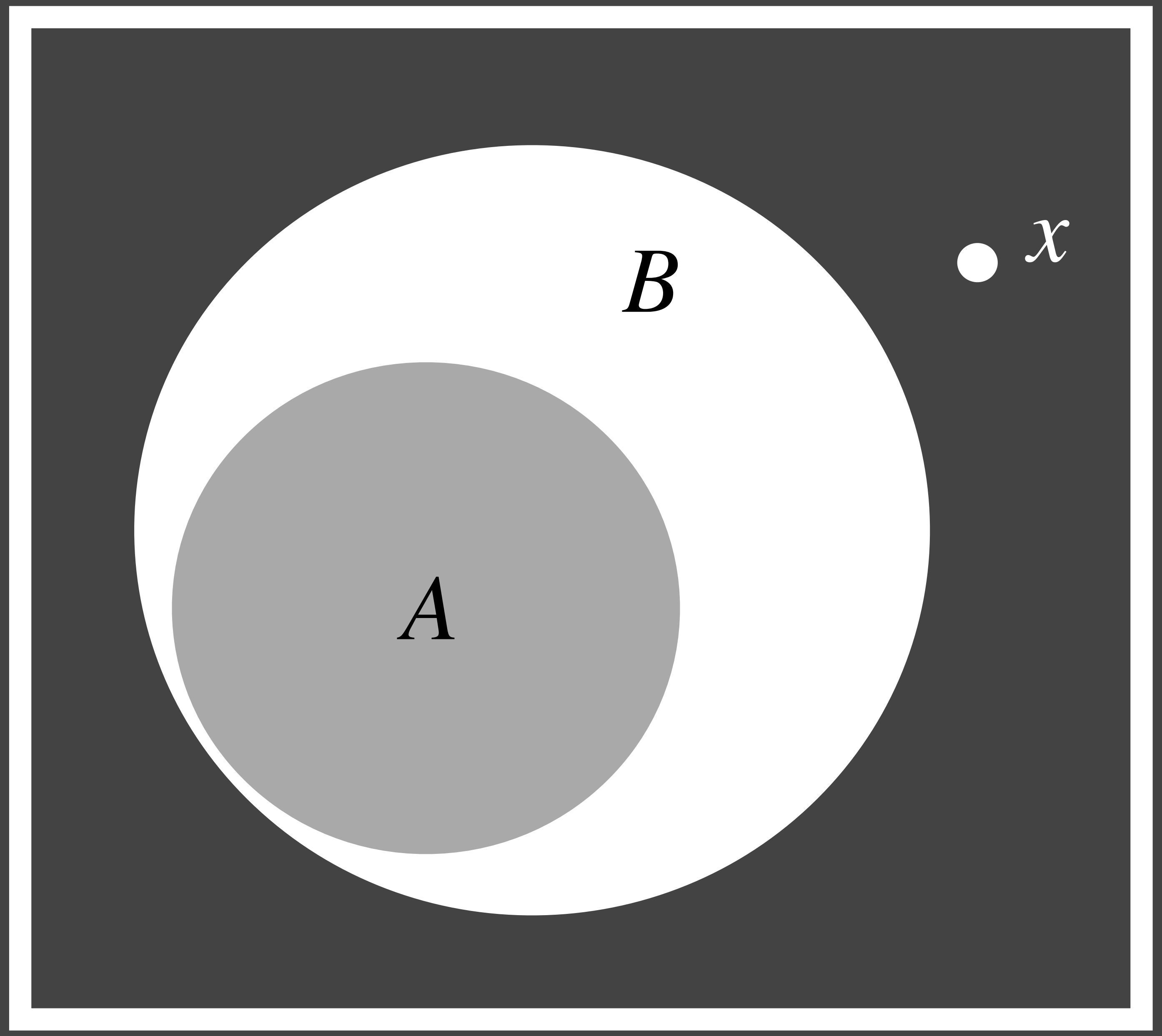


Submodularity

a.k.a. "Decreasing Marginal Returns!"

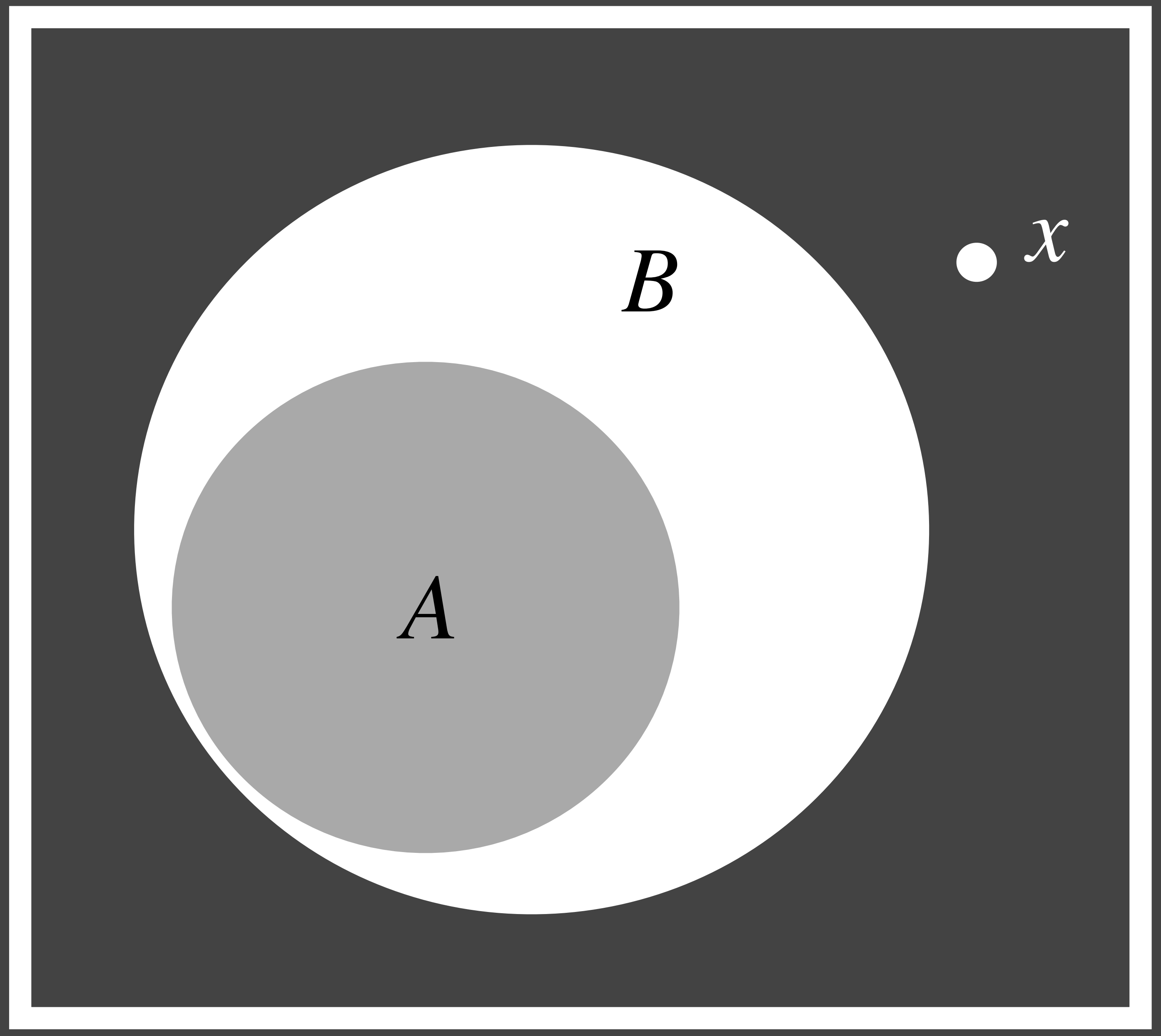
Definition: f is ***submodular*** *if,* $\forall A \subseteq B, x \notin B,$

$$f(A + x) - f(A) \geq f(B + x) - f(B)$$



Submodularity

a.k.a. "Decreasing Marginal Returns!"



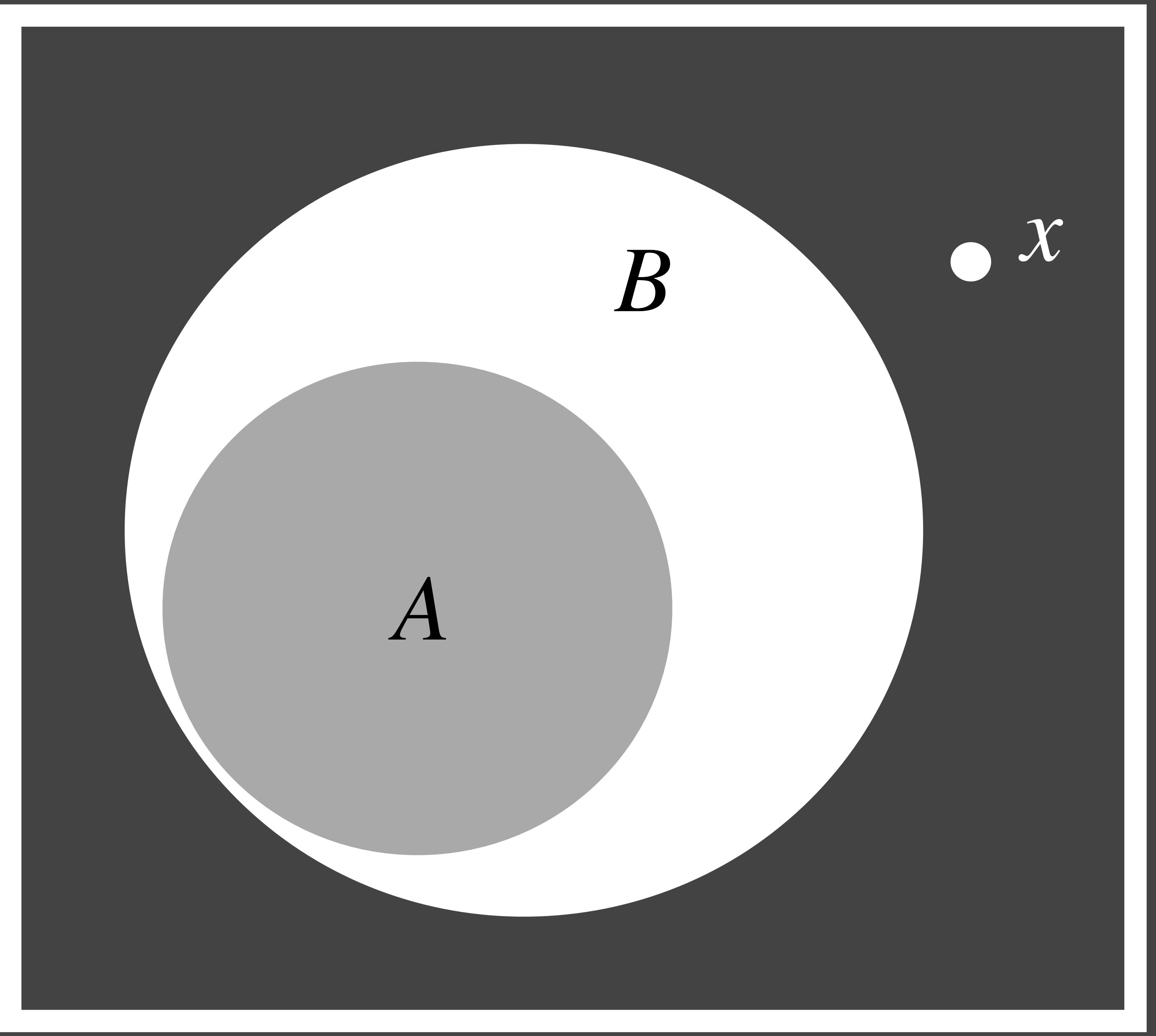
Definition: f is ***submodular*** *if,* $\forall A \subseteq B, x \notin B,$

$$f(A + x) - f(A) \geq f(B + x) - f(B)$$

i.e. $f(x \mid A) \geq f(x \mid B)$

Submodularity

a.k.a. "Decreasing Marginal Returns!"



<u>Definition</u>: f is ***submodular*** *if,* $\forall A \subseteq B, x \notin B,$

$$f(A + x) - f(A) \geq f(B + x) - f(B)$$

i.e. $\quad f(x \mid A) \geq f(x \mid B)$

$$f(🍕 \mid 🥕) \geq f(🍕 \mid 🥕, 🍩)$$

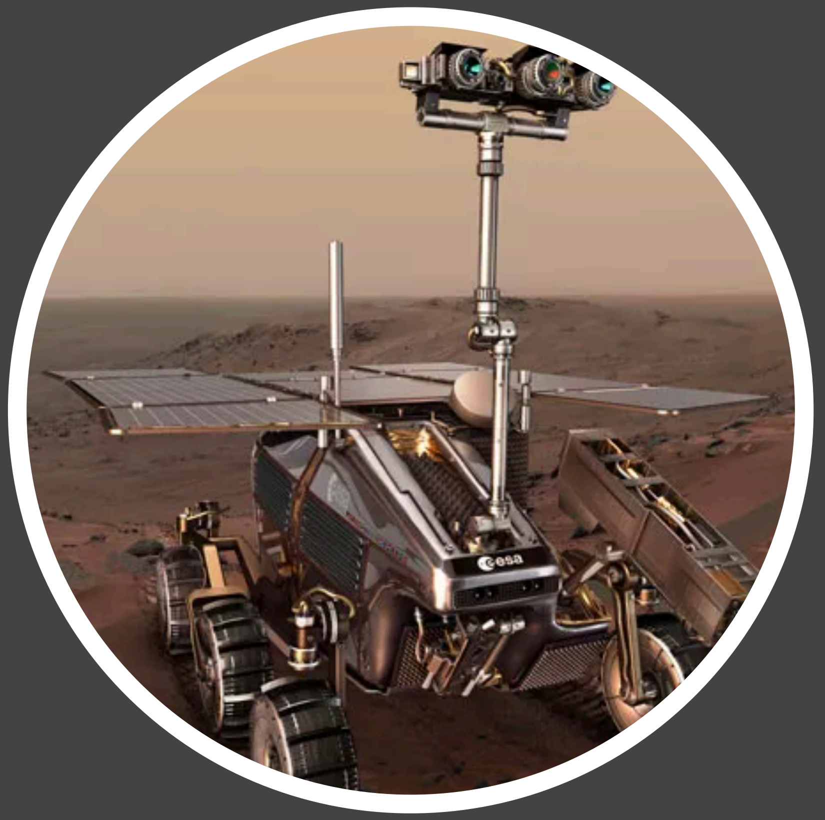
Why care about Submodular Cover?

Why care about Submodular Cover?

1. Highly expressive! Examples of Submodular Cover:

Why care about Submodular Cover?

1. Highly expressive! Examples of Submodular Cover:



Robot Exploration

Why care about Submodular Cover?

1. Highly expressive! Examples of Submodular Cover:

Robot Exploration

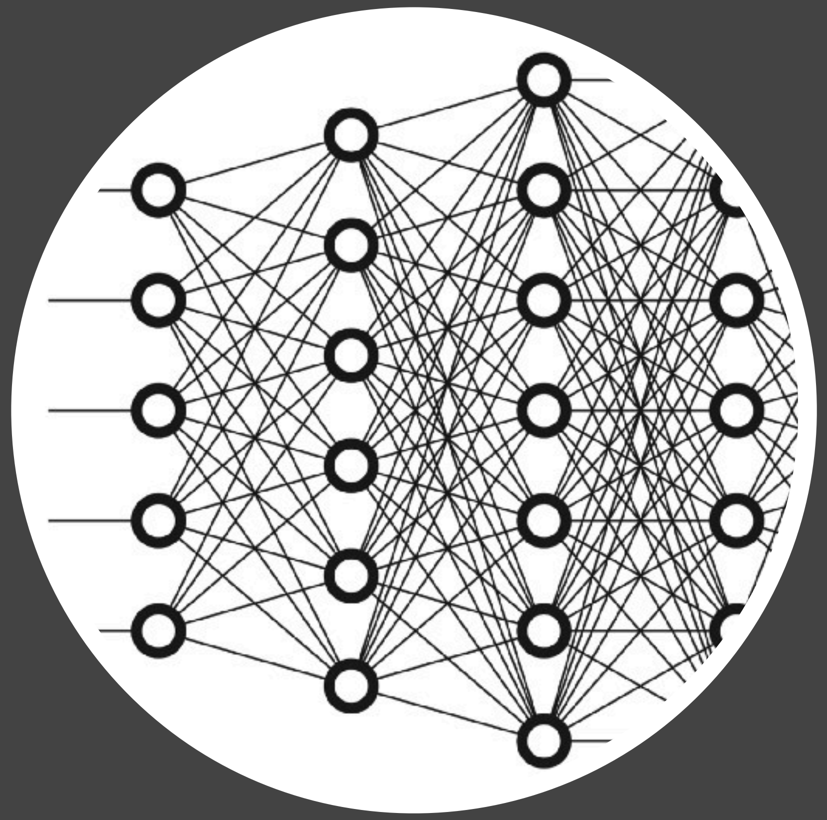
Influence Maximization

Why care about Submodular Cover?

1. Highly expressive! Examples of Submodular Cover:

Robot Exploration

Influence Maximization



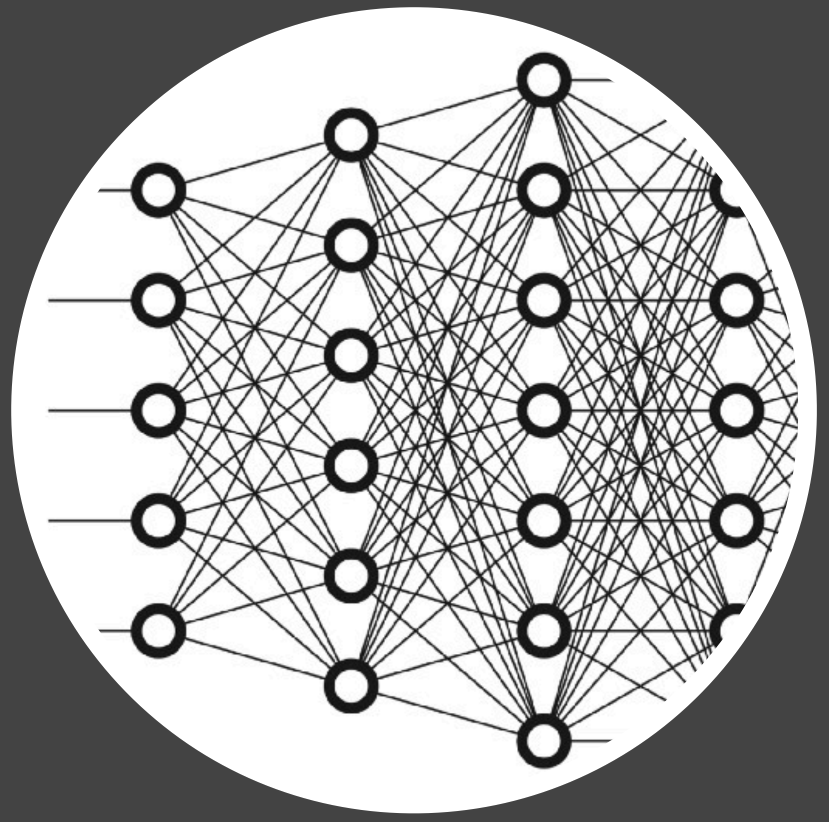
Feature Selection

Why care about Submodular Cover?

1. Highly expressive! Examples of Submodular Cover:

Robot
Exploration

Influence
Maximization



Feature
Selection

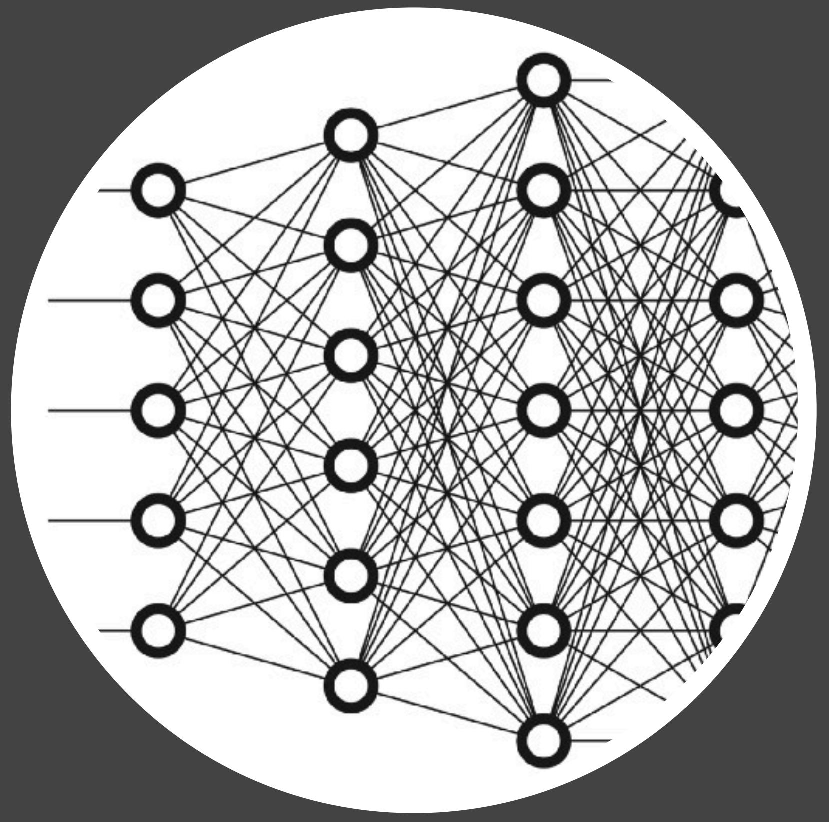
Document
Summarization

Why care about Submodular Cover?

1. Highly expressive! Examples of Submodular Cover:

Robot Exploration

Influence Maximization



Feature Selection

Document Summarization

Resource allocation

Why care about Submodular Cover?

Popular to reduce to Submodular Cover!
[Goyal+ 13][Loukides Gwadera 16][Zheng+ 17][Andreev+ 09][Lee+ 13][Lukovszki+ 18][Poularakis+ 17][Krause+ 08][Kortsarz Nutov 15][Jorgensen+ 17][Chen+ 18][Beinhofer+ 13][Tzoumas+ 16][Tong+ 17][Liu+ 16][Mafuta Walingo 16][Yang+ 15][Rahimian Preciado 15][Izumi+ 10][Wu+ 15], [Shin+ 23], [Gong+ 23], [Li+ 23], [Coester, Naor, L., Talmon 22] etc…

Why care about Submodular Cover?

Popular to reduce to Submodular Cover!
[Goyal+ 13][Loukides Gwadera 16][Zheng+ 17][Andreev+ 09][Lee+ 13][Lukovszki+ 18][Poularakis+ 17][Krause+ 08][Kortsarz Nutov 15][Jorgensen+ 17][Chen+ 18][Beinhofer+ 13][Tzoumas+ 16][Tong+ 17][Liu+ 16][Mafuta Walingo 16][Yang+ 15][Rahimian Preciado 15][Izumi+ 10][Wu+ 15], [Shin+ 23], [Gong+ 23], [Li+ 23], [Coester, Naor, L., Talmon 22] etc…

Porting submod cover to uncertain settings automatically ports all applications!

Why care about Submodular Cover?

Popular to reduce to Submodular Cover!
[Goyal+ 13][Loukides Gwadera 16][Zheng+ 17][Andreev+ 09][Lee+ 13][Lukovszki+ 18][Poularakis+ 17][Krause+ 08][Kortsarz Nutov 15][Jorgensen+ 17][Chen+ 18][Beinhofer+ 13][Tzoumas+ 16][Tong+ 17][Liu+ 16][Mafuta Walingo 16][Yang+ 15][Rahimian Preciado 15][Izumi+ 10][Wu+ 15], [Shin+ 23], [Gong+ 23], [Li+ 23], [Coester, Naor, L., Talmon 22] etc…

Porting submod cover to uncertain settings automatically ports all applications!

2. Fast algos get good approximation: $O(\log n)$ [Wolsey 82]

Why care about Submodular Cover?

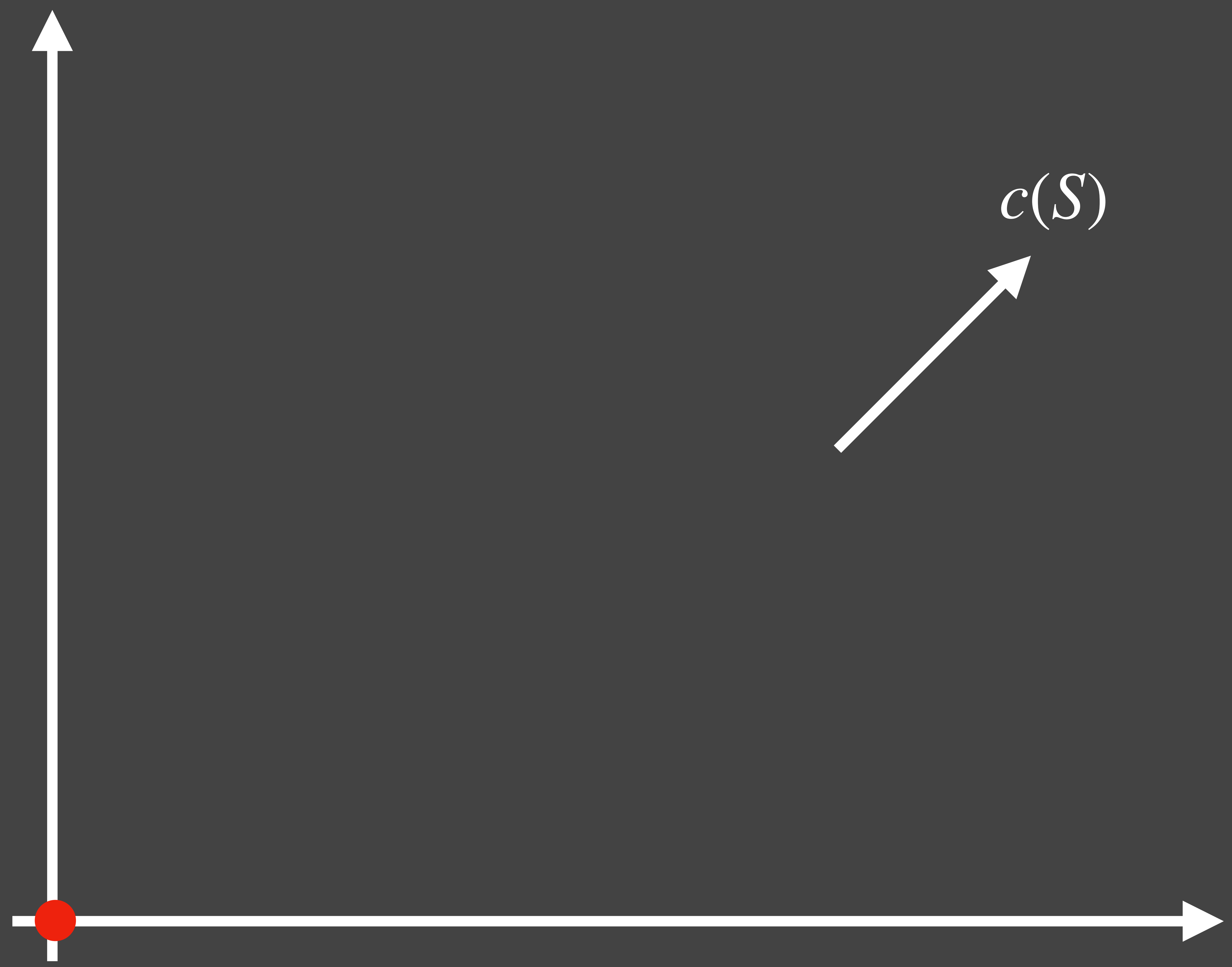
Popular to reduce to Submodular Cover!
[Goyal+ 13][Loukides Gwadera 16][Zheng+ 17][Andreev+ 09][Lee+ 13][Lukovszki+ 18][Poularakis+ 17][Krause+ 08][Kortsarz Nutov 15][Jorgensen+ 17][Chen+ 18][Beinhofer+ 13][Tzoumas+ 16][Tong+ 17][Liu+ 16][Mafuta Walingo 16][Yang+ 15][Rahimian Preciado 15][Izumi+ 10][Wu+ 15], [Shin+ 23], [Gong+ 23], [Li+ 23], [Coester, Naor, L., Talmon 22] etc…

Porting submod cover to uncertain settings automatically ports all applications!

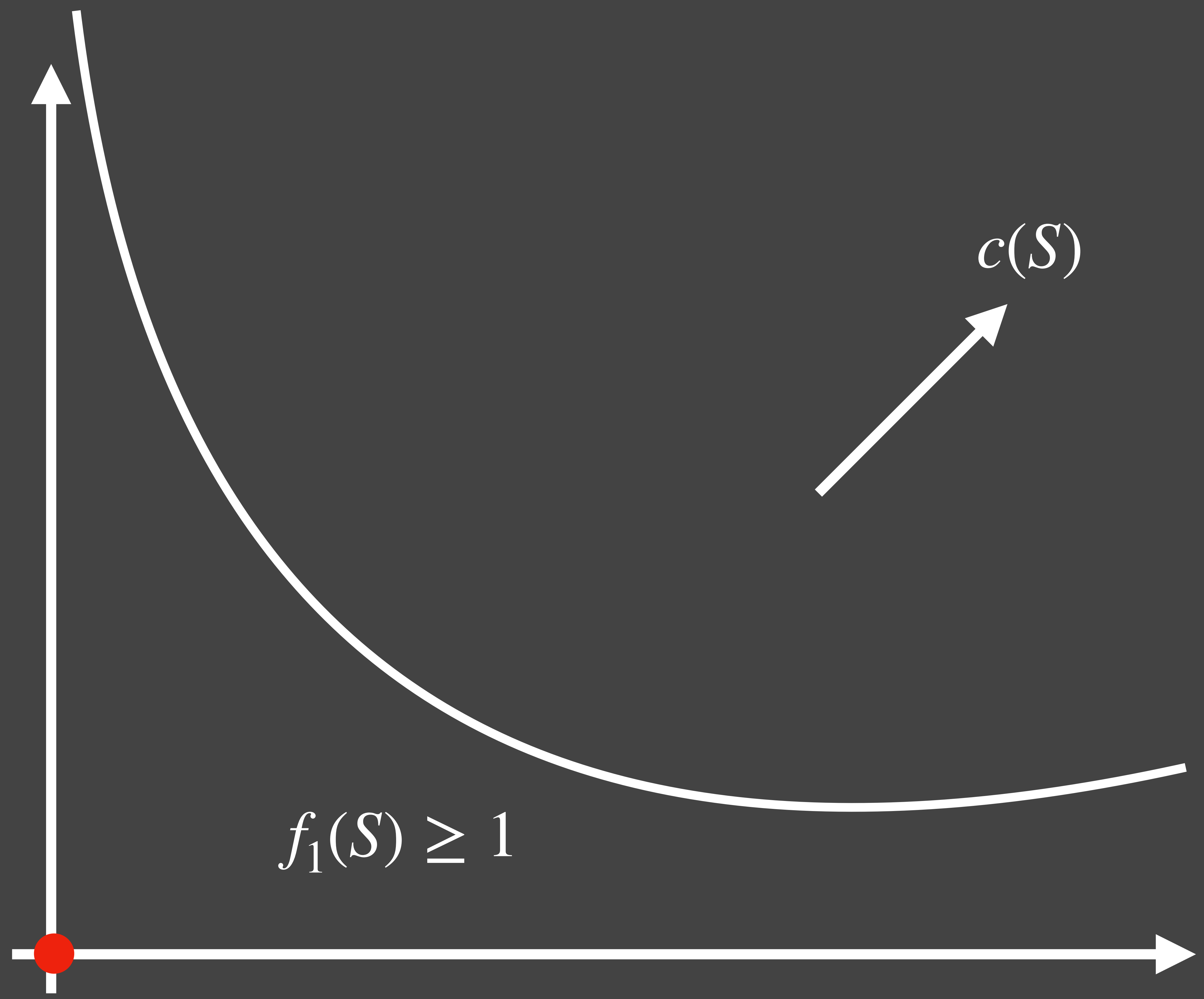
2. Fast algos get good approximation: $O(\log n)$ [Wolsey 82]

Punchline: Sweet spot between generality and tractability!

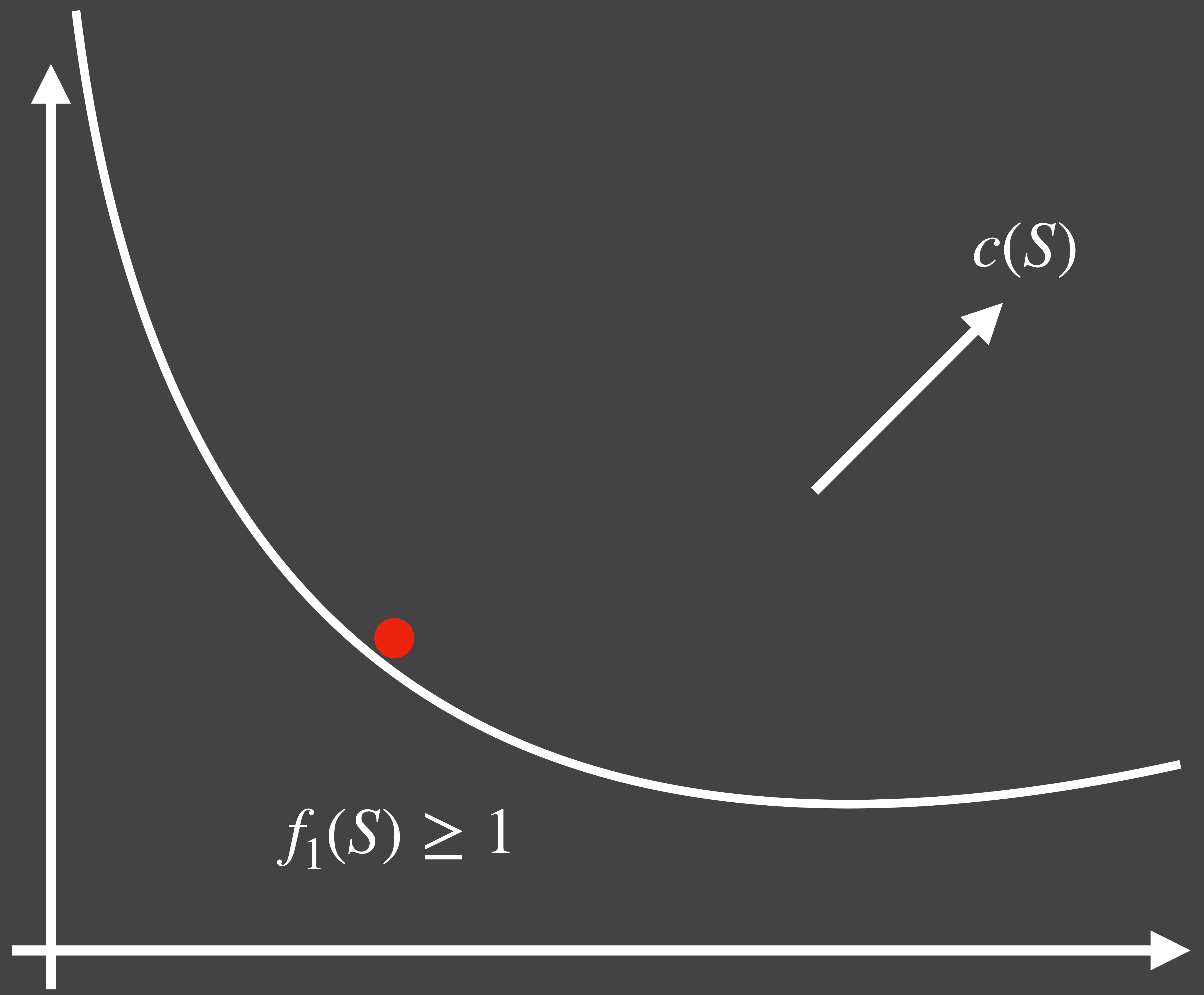
Online Submodular Cover [Gupta L. SODA 20]



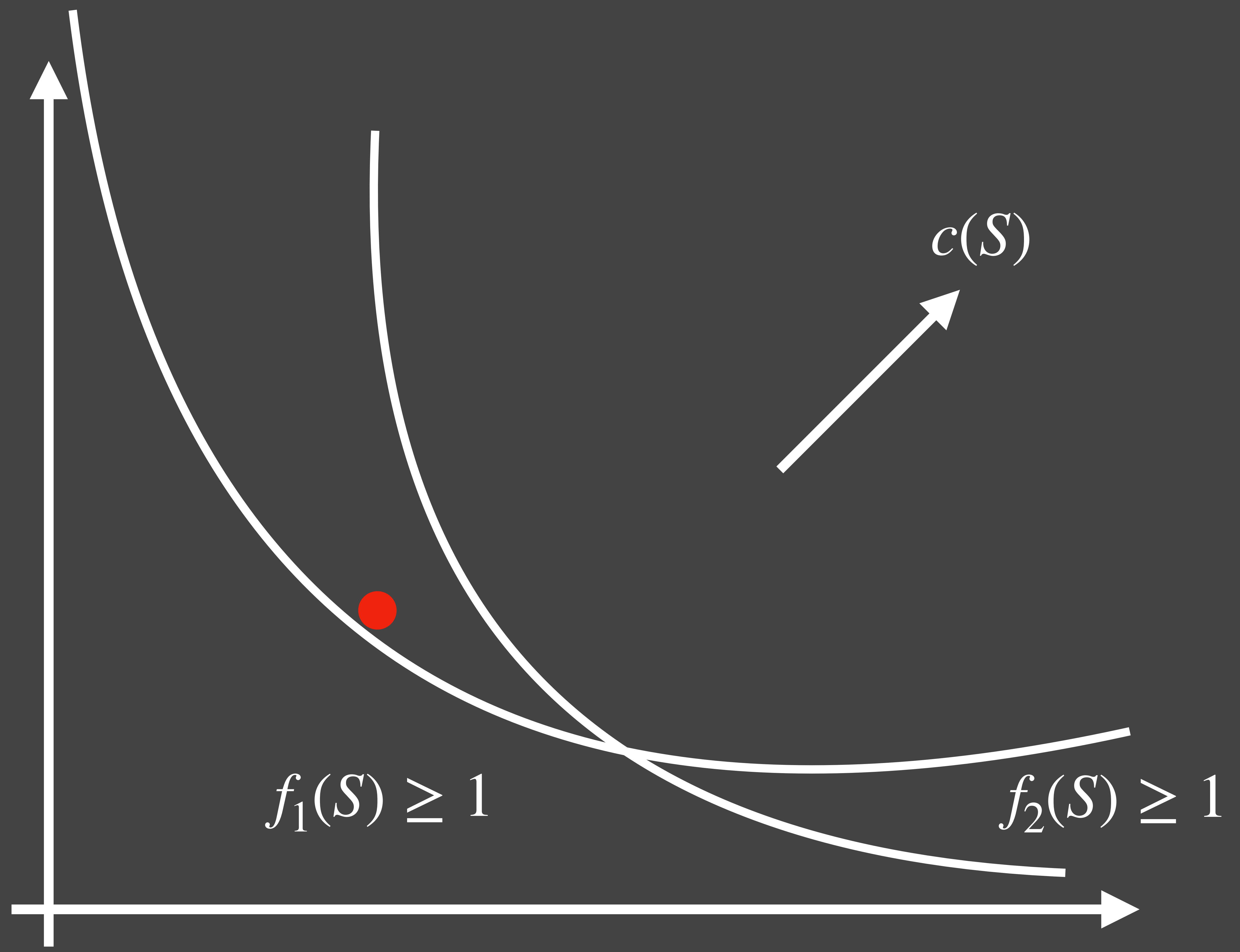
Online Submodular Cover [Gupta L. SODA 20]



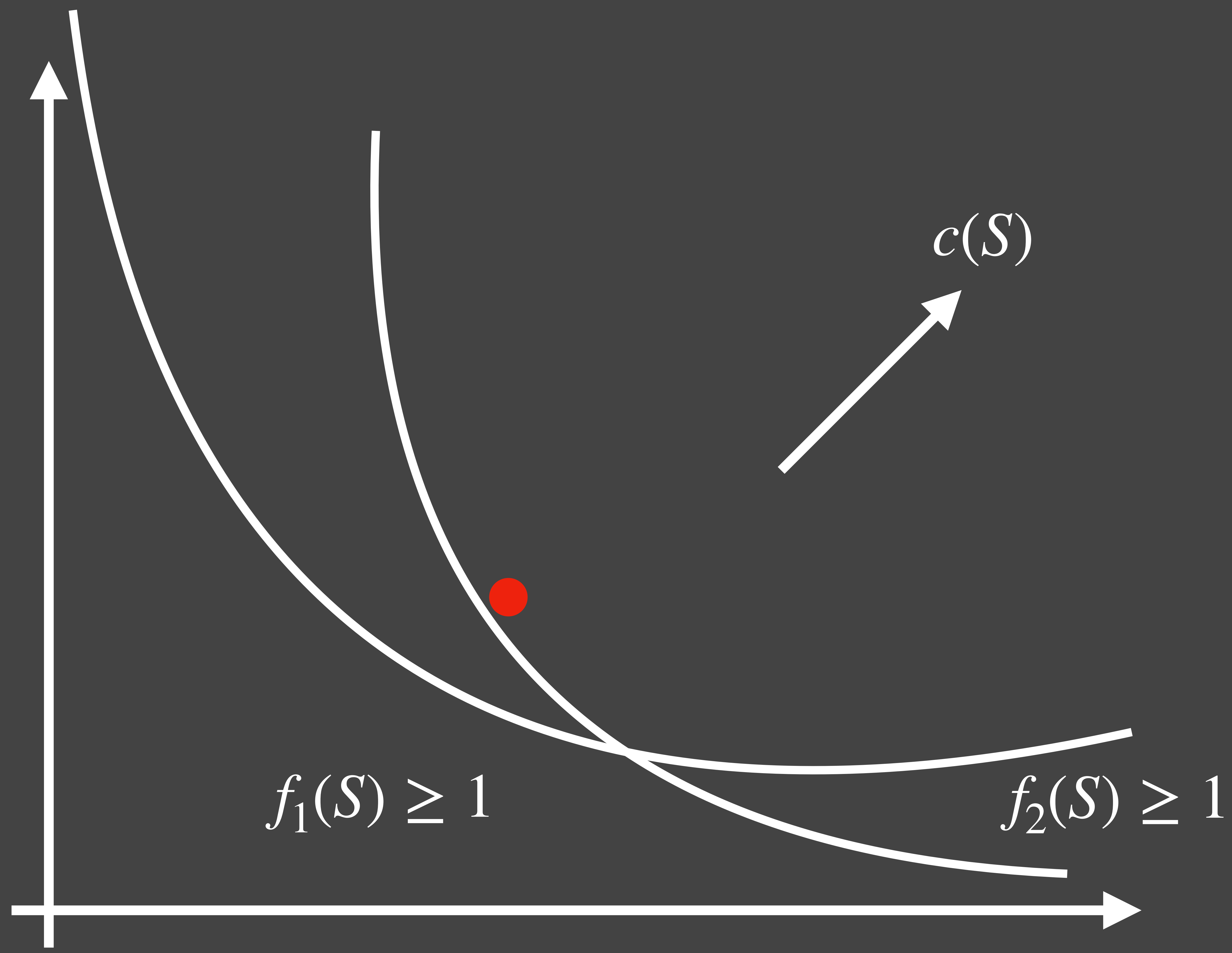
Online Submodular Cover [Gupta L. SODA 20]



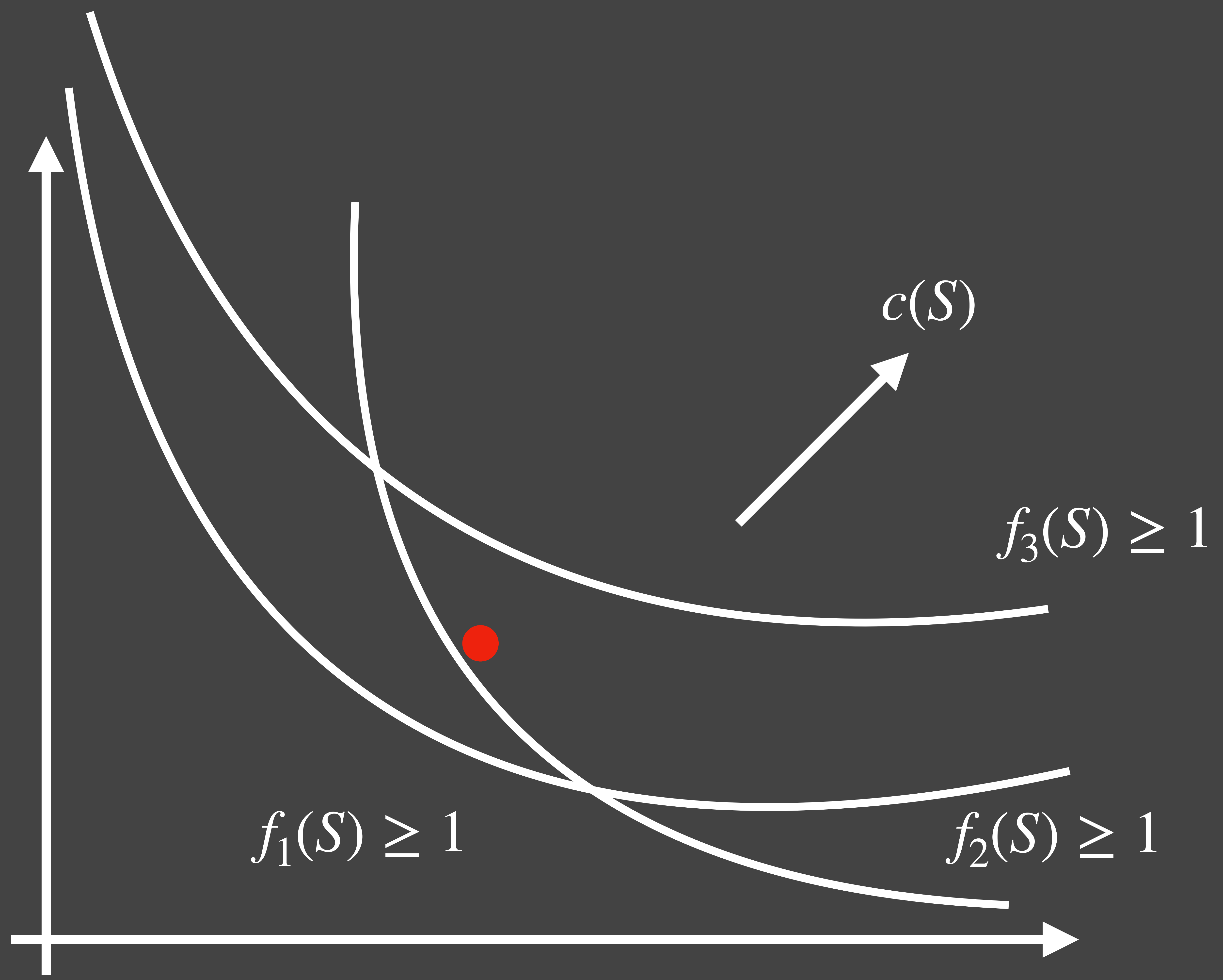
Online Submodular Cover [Gupta L. SODA 20]



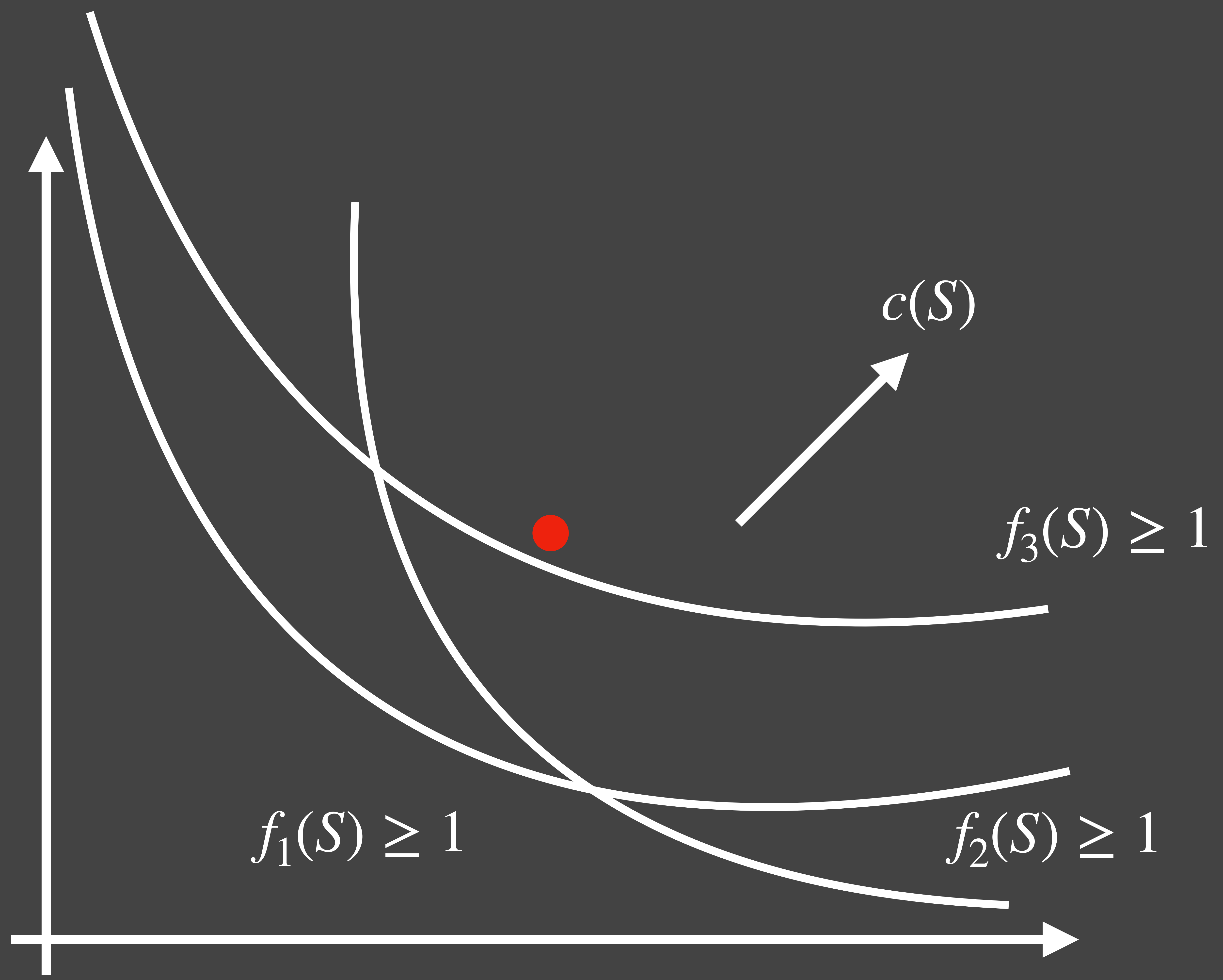
Online Submodular Cover [Gupta L. SODA 20]



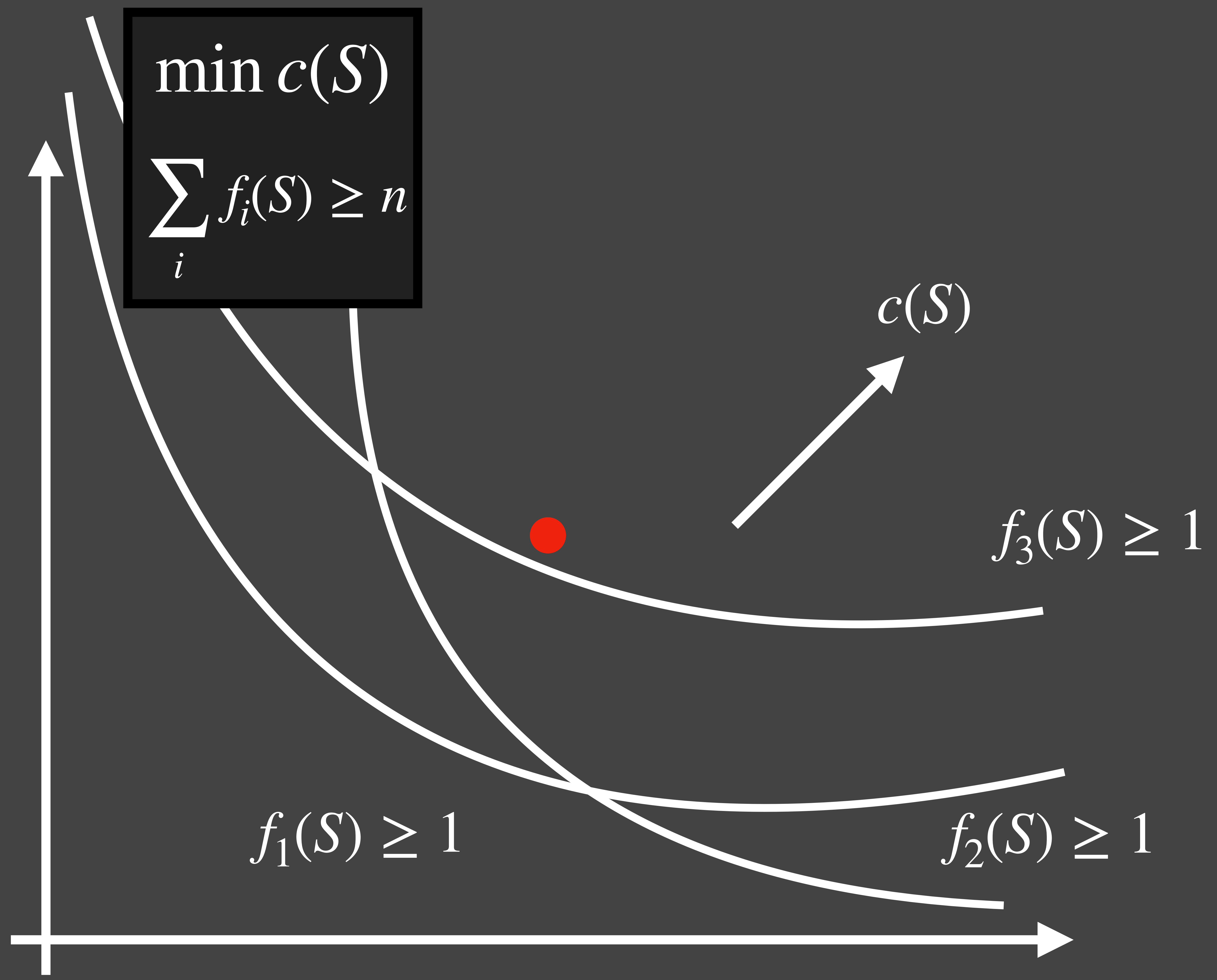
Online Submodular Cover [Gupta L. SODA 20]



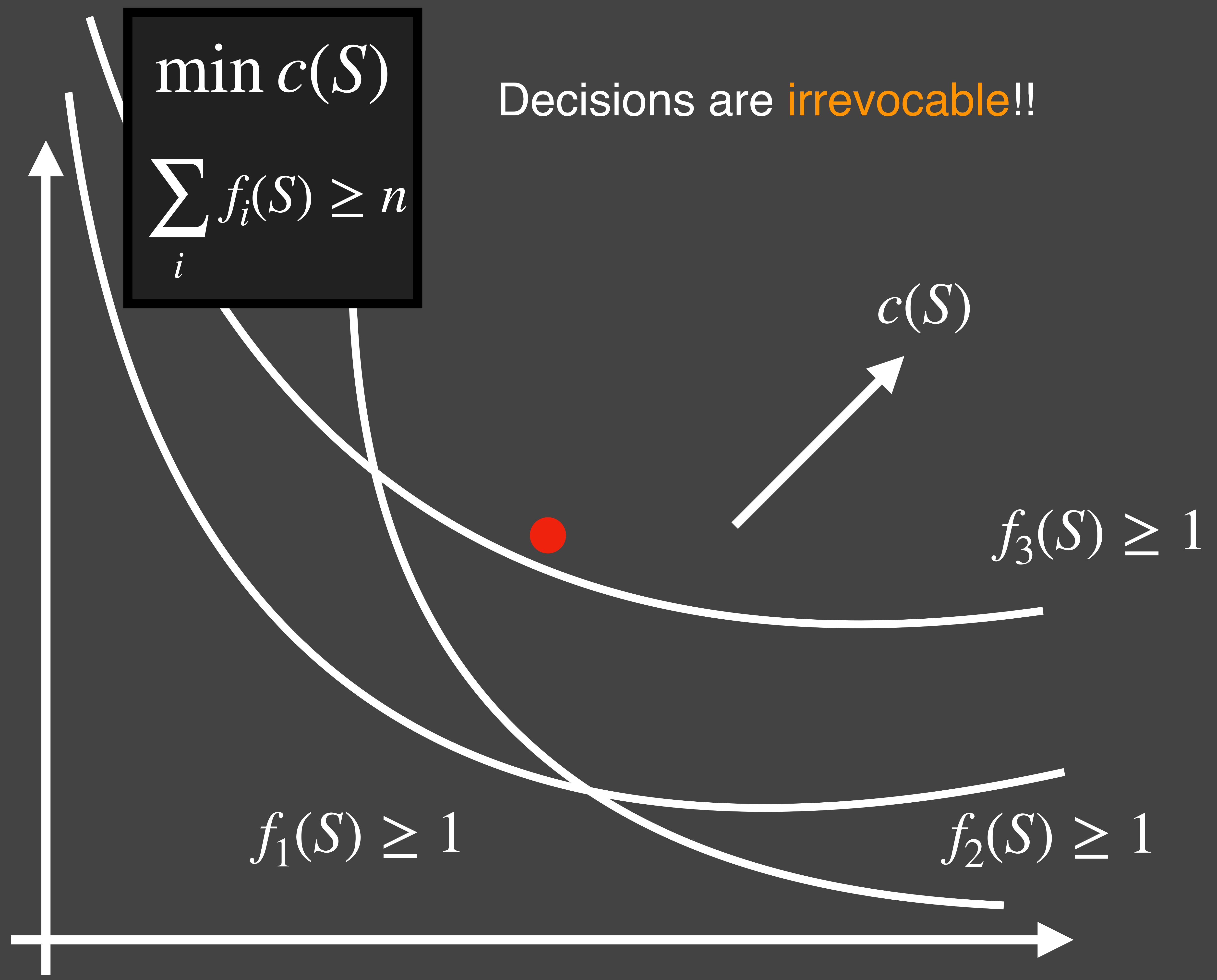
Online Submodular Cover [Gupta L. SODA 20]



Online Submodular Cover [Gupta L. SODA 20]



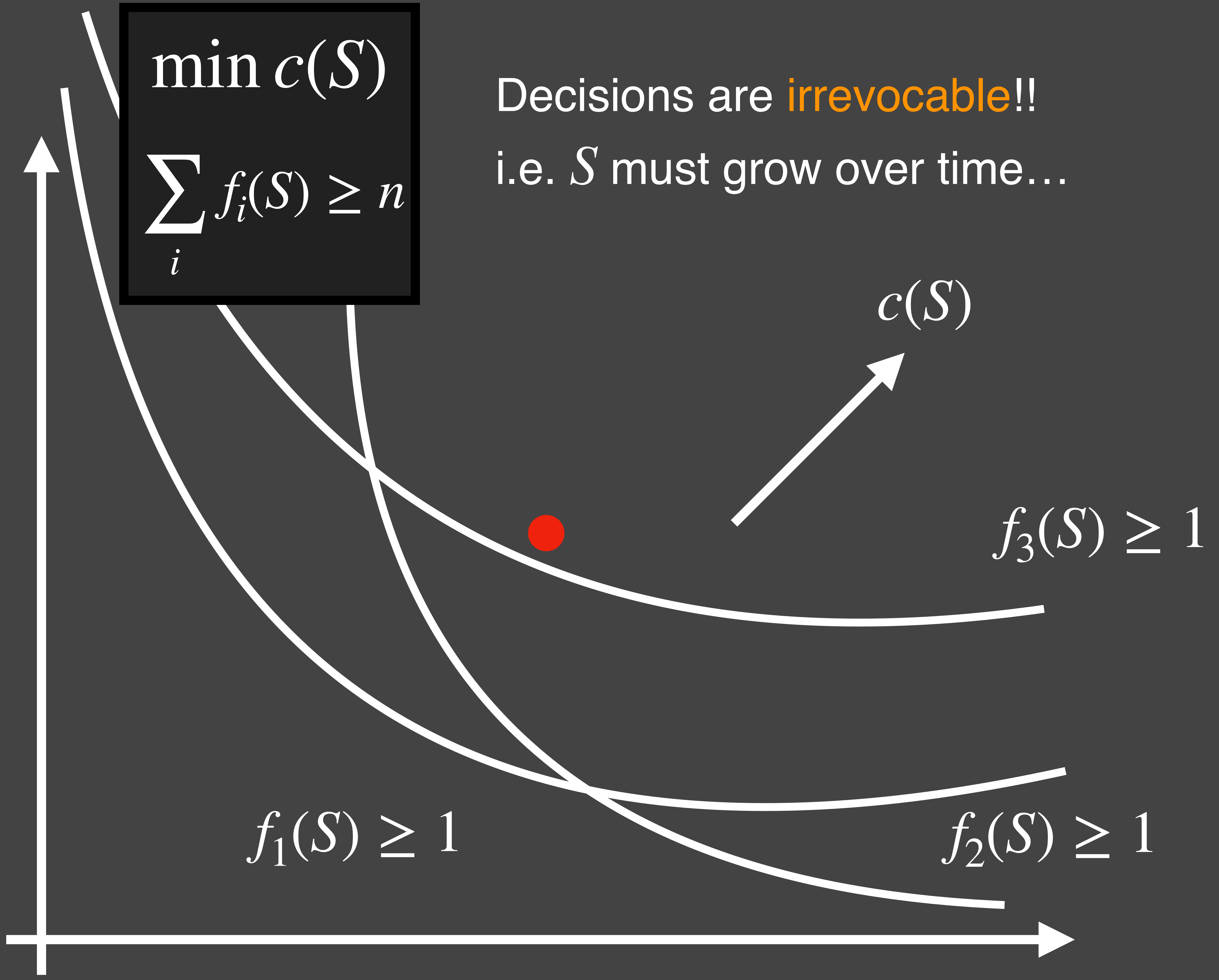
Online Submodular Cover [Gupta L. SODA 20]



Decisions are irrevocable!!

Online Submodular Cover

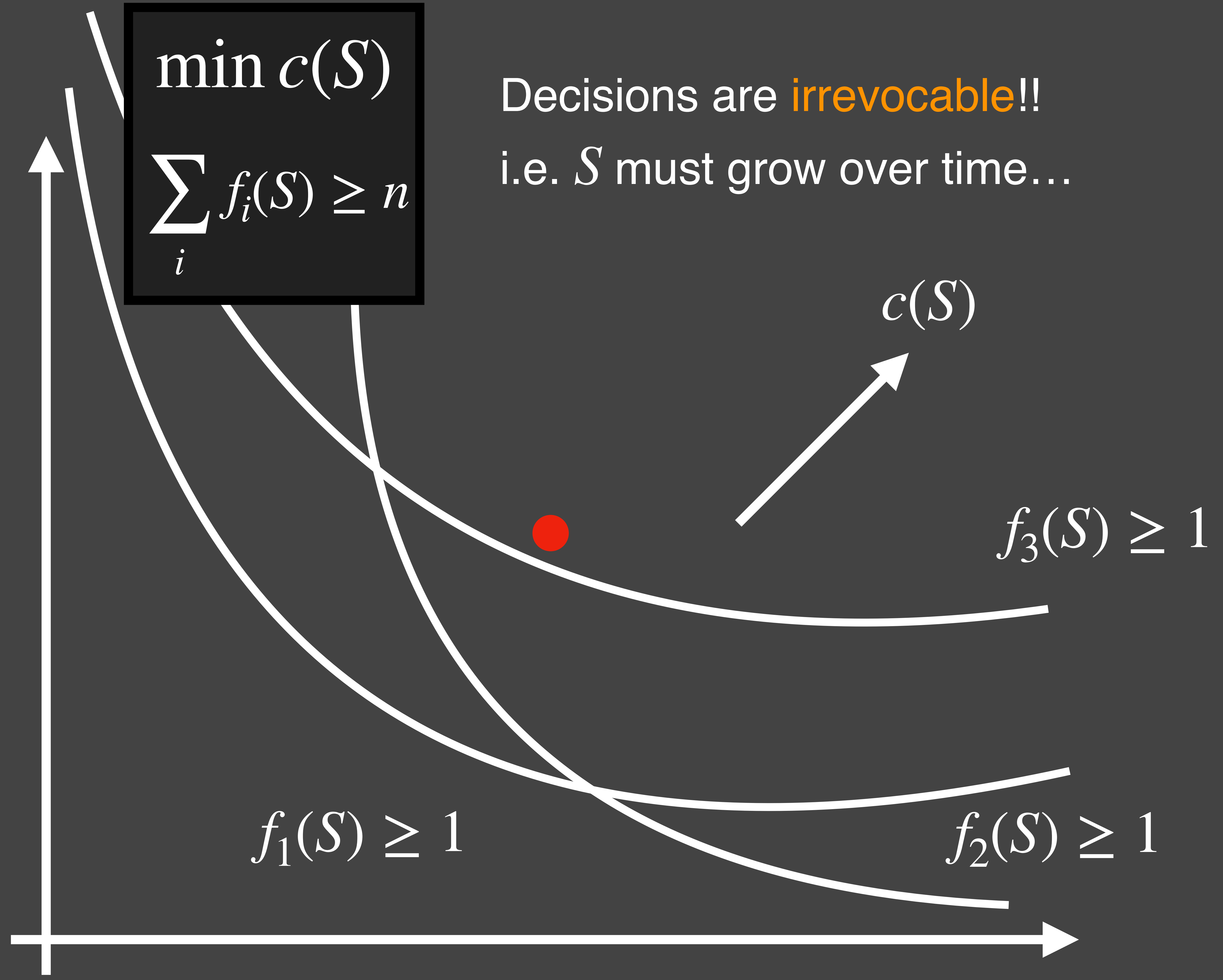
[Gupta L. SODA 20]



Decisions are irrevocable!!

i.e. S must grow over time…

Online Submodular Cover [Gupta L. SODA 20]



Decisions are irrevocable!!

i.e. S must grow over time…

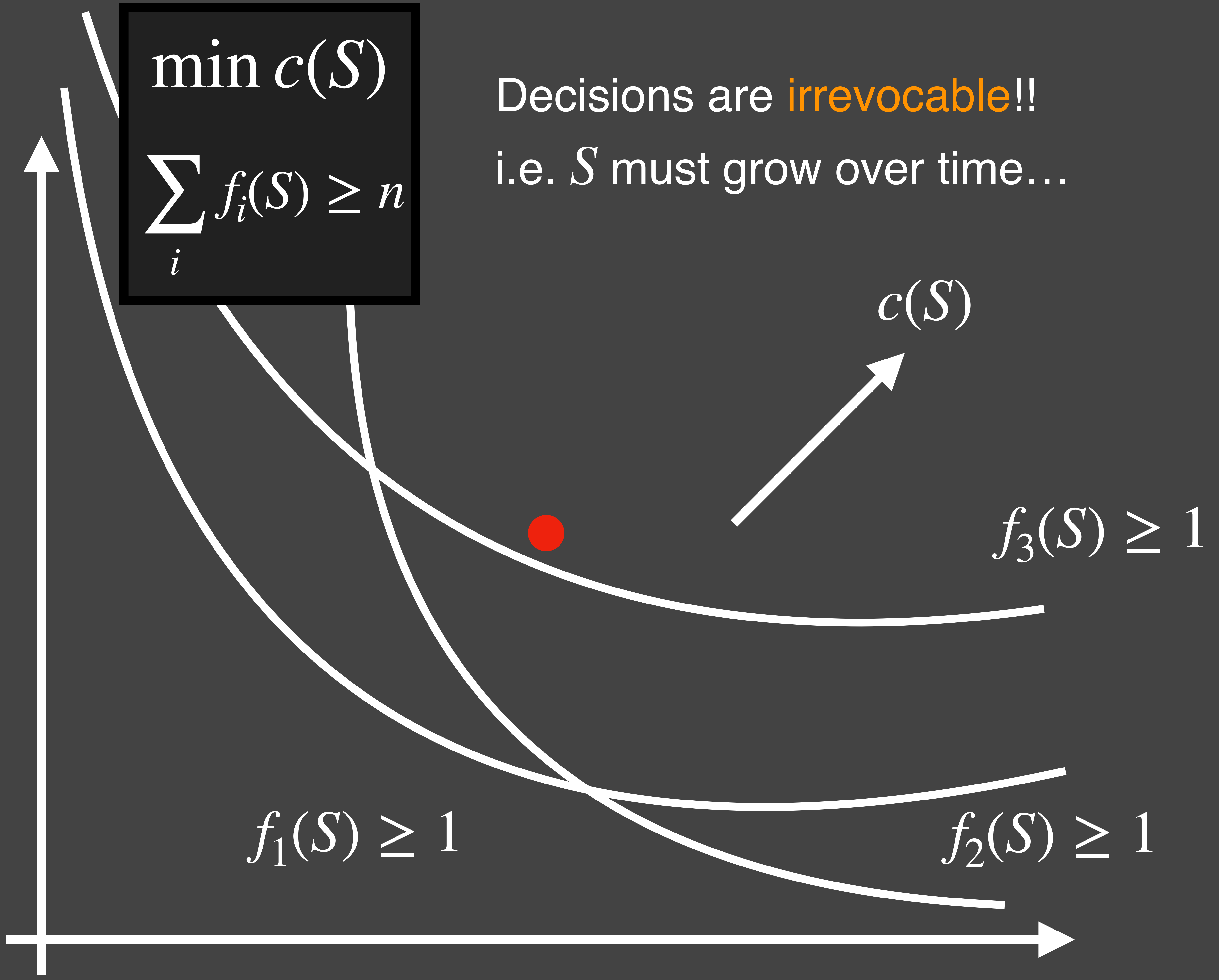
Theorem [Gupta L. SODA 20]:

Polynomial time algo for Online Submod Cover with approximation $O(\log^2 n)$.

Online Submodular Cover [Gupta L. SODA 20]

Decisions are irrevocable!!

i.e. S must grow over time…

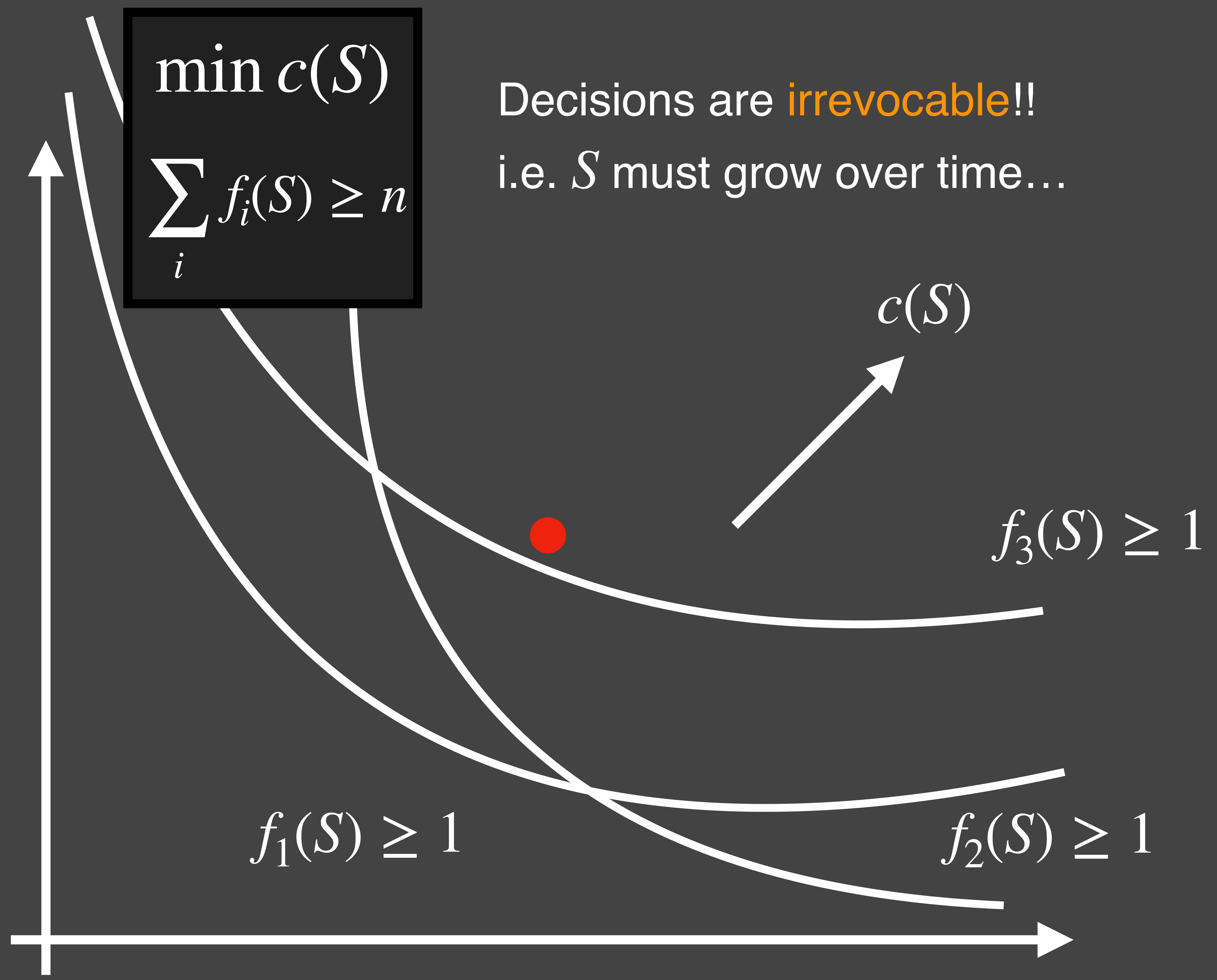


Theorem [Gupta L. SODA 20]:

Polynomial time algo for Online Submod Cover with approximation $O(\log^2 n)$.

Optimal!

Online Submodular Cover [Gupta L. SODA 20]



Theorem [Gupta L. SODA 20]:

Polynomial time algo for Online Submod Cover with approximation $O(\log^2 n)$.

Optimal!

Technical Ingredient:
RoundOrSeparate for LP relaxation of Submodular Cover & generalization of Mutual Information!

Online Submodular Cover [Gupta L. SODA 20]

Online Set Cover
$O(\log^2 n)$

Submodular Cover
$O(\log n)$

Set Cover
$O(\log n)$

Online Submodular Cover [Gupta L. SODA 20]

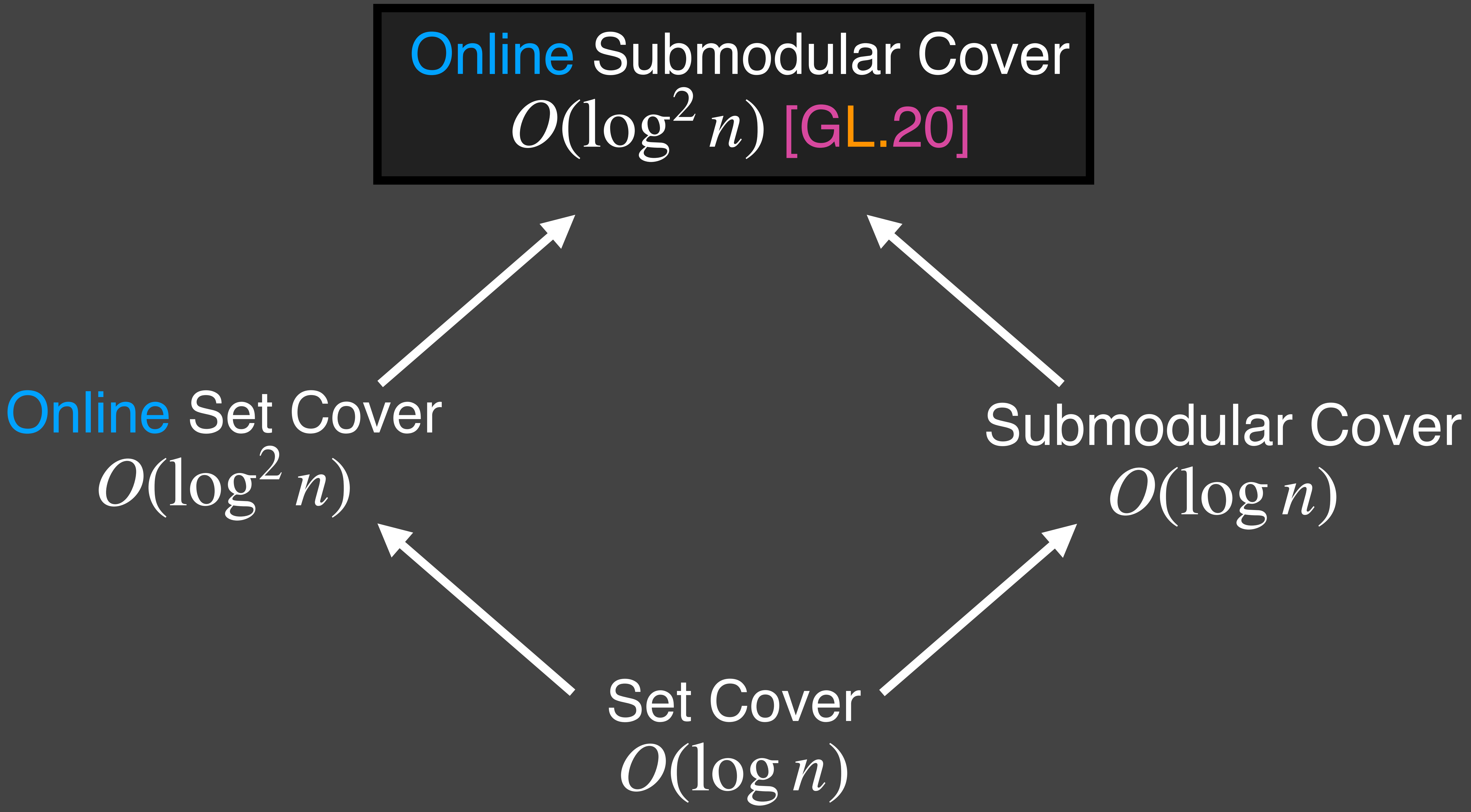
Online Submodular Cover
$O(\log^2 n)$ [GL.20]

Online Set Cover
$O(\log^2 n)$

Submodular Cover
$O(\log n)$

Set Cover
$O(\log n)$

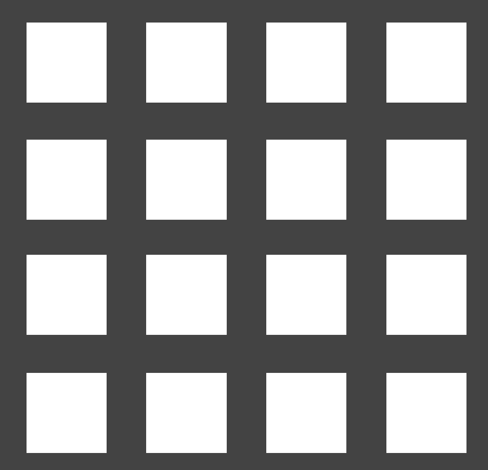
Online Submodular Cover [Gupta L. SODA 20]



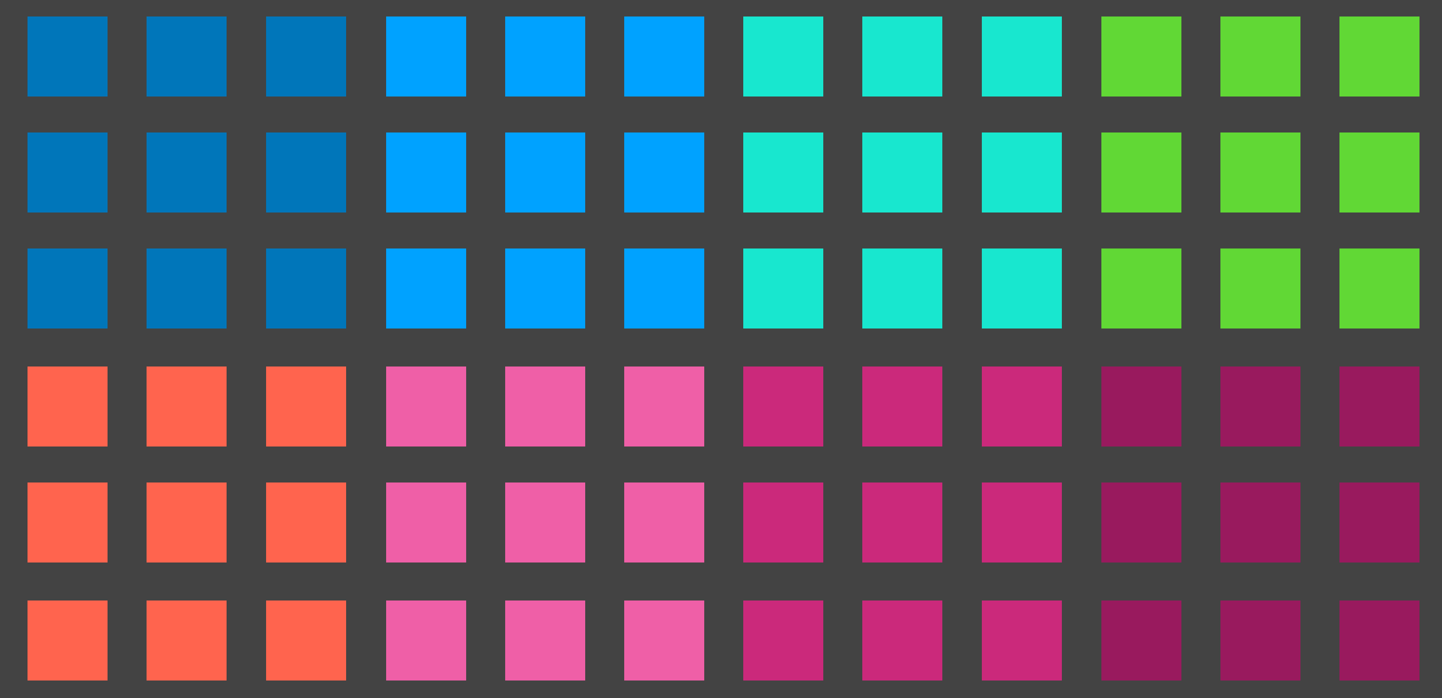
Best of both worlds: modeling power of Submodular Cover + Online.

Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

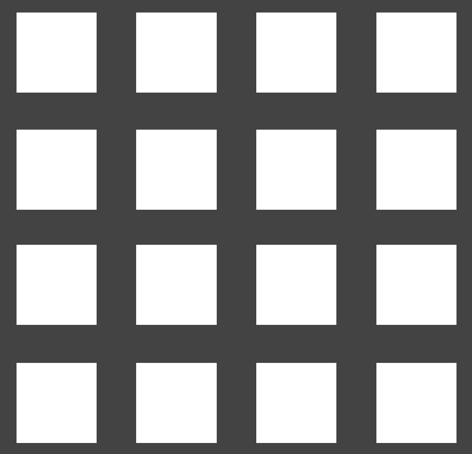


n total pages, **divided into blocks**

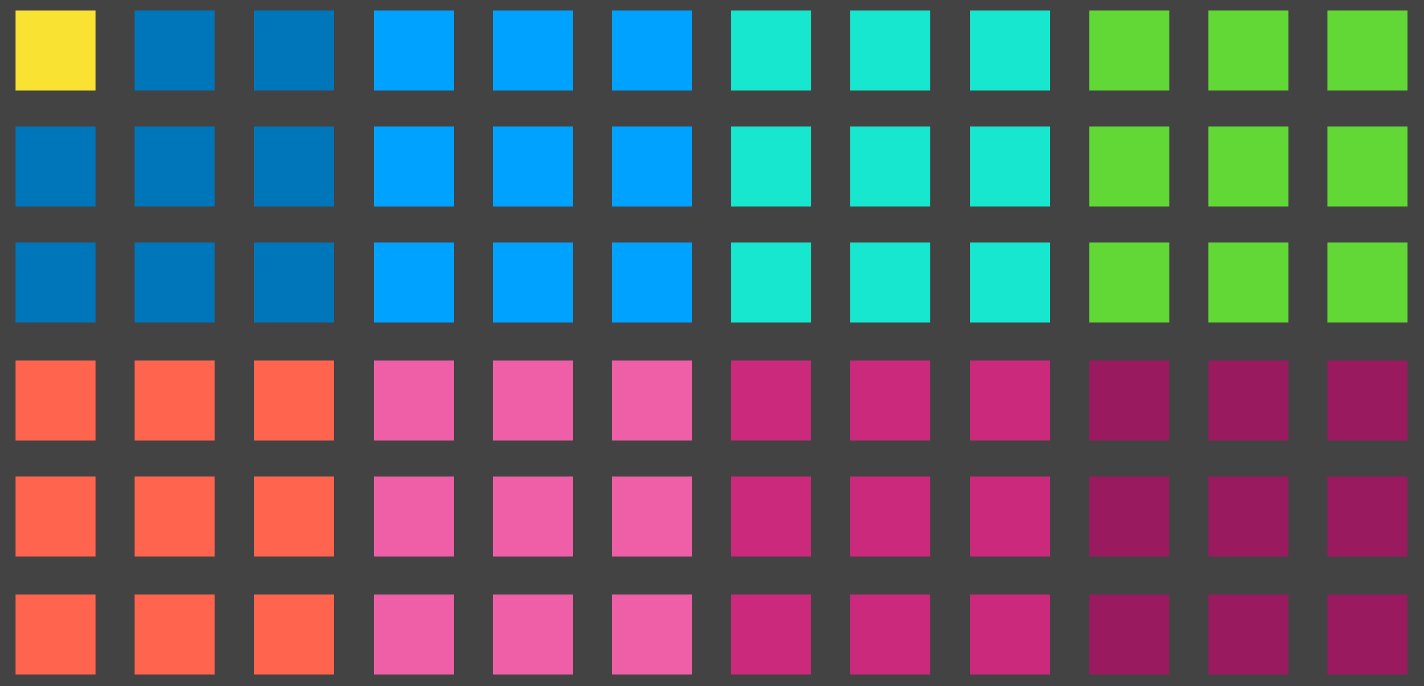


Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

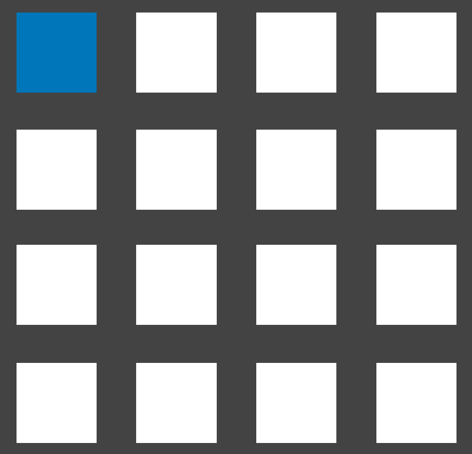


n total pages, **divided into blocks**

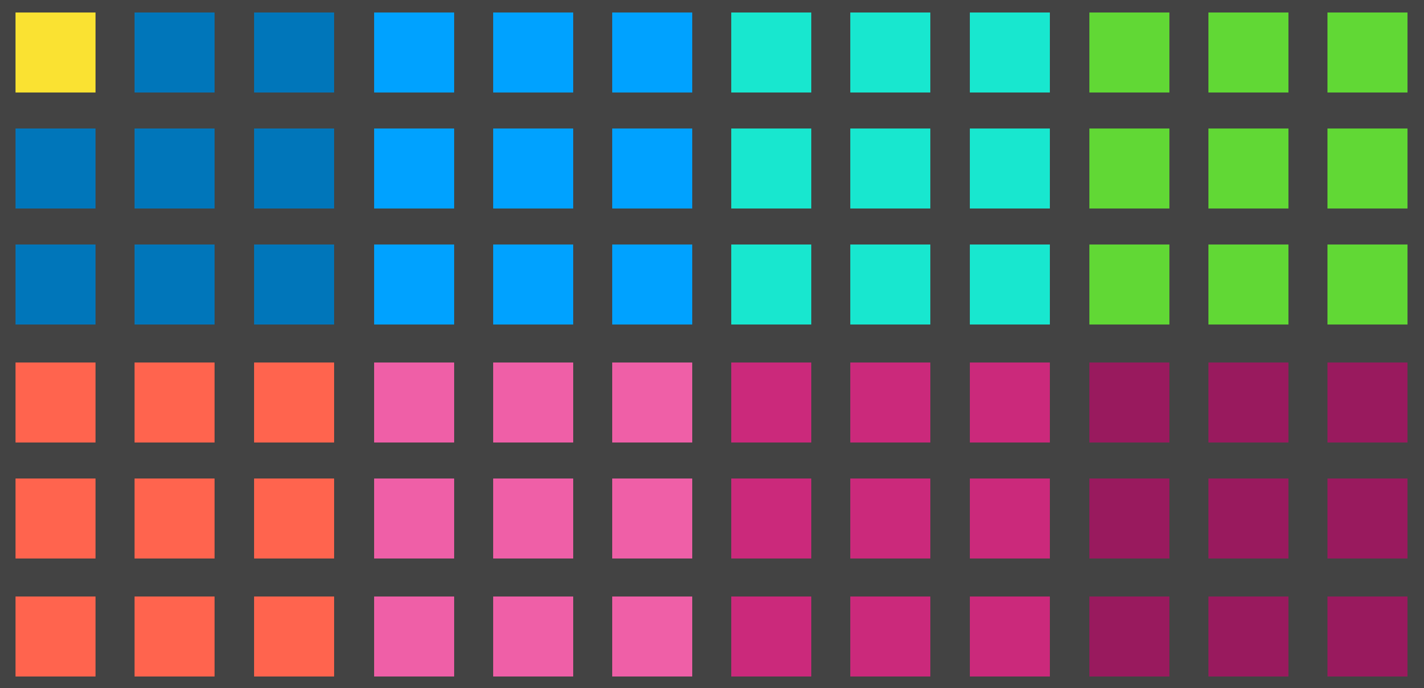


Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

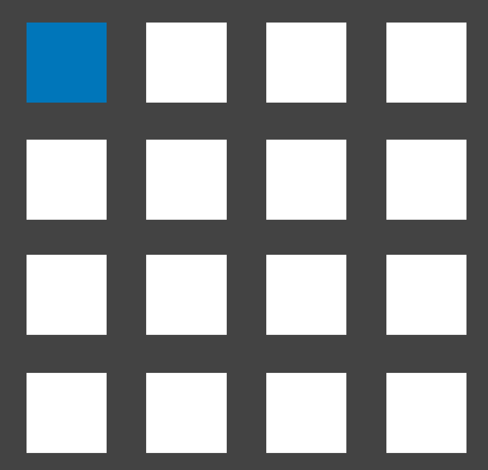


n total pages, **divided into blocks**

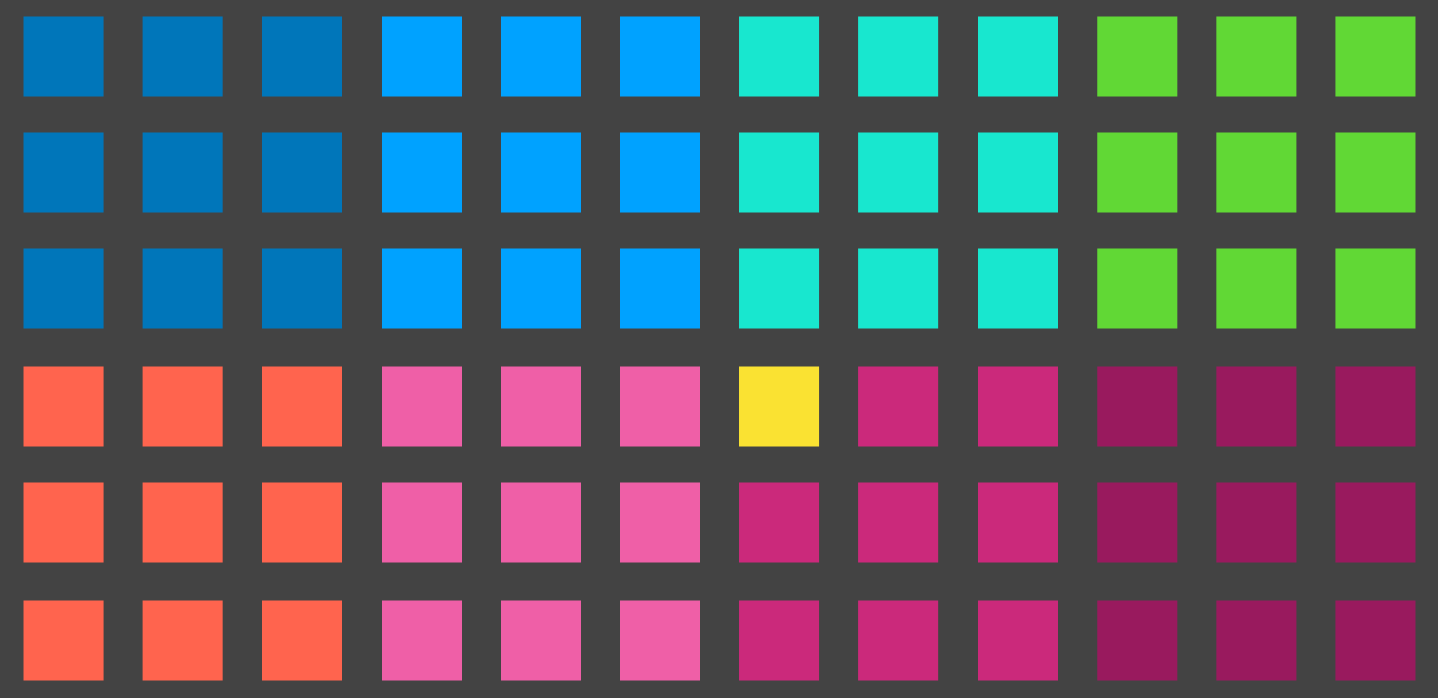


Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

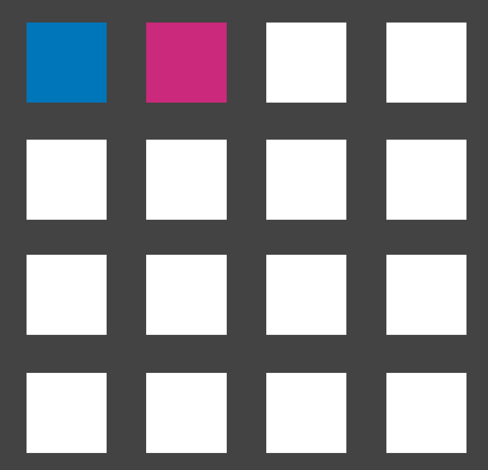


n total pages, **divided into blocks**

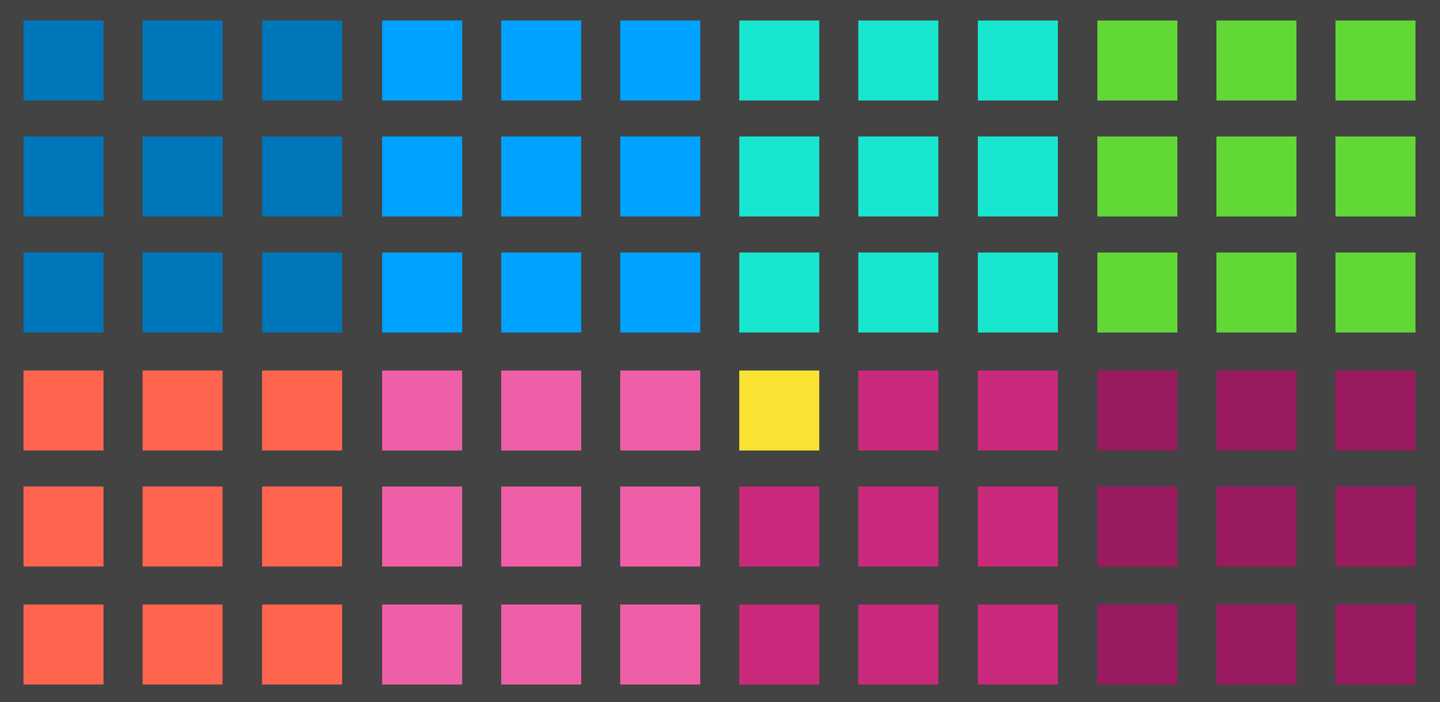


Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

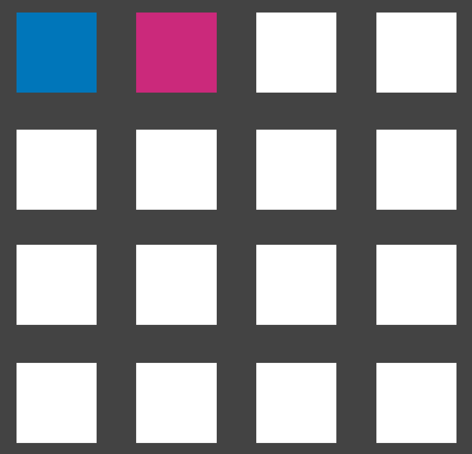


n total pages, **divided into blocks**

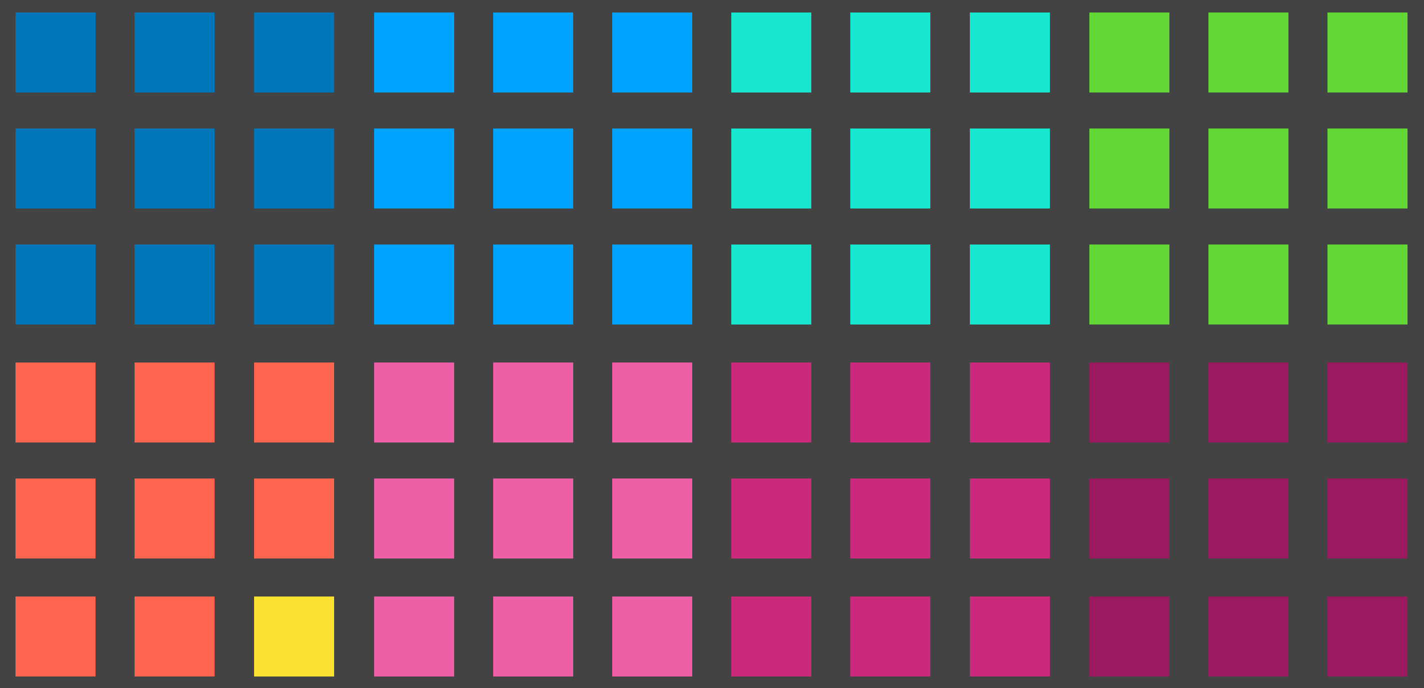


Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

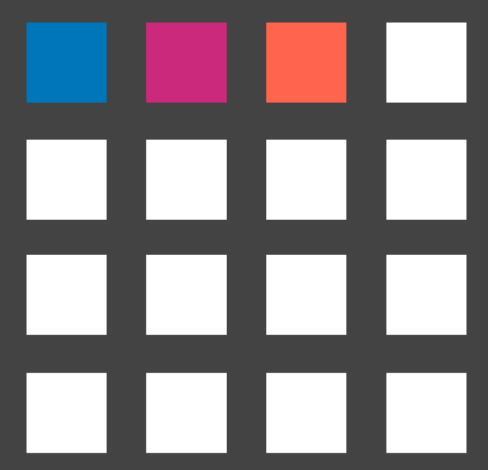


n total pages, **divided into blocks**

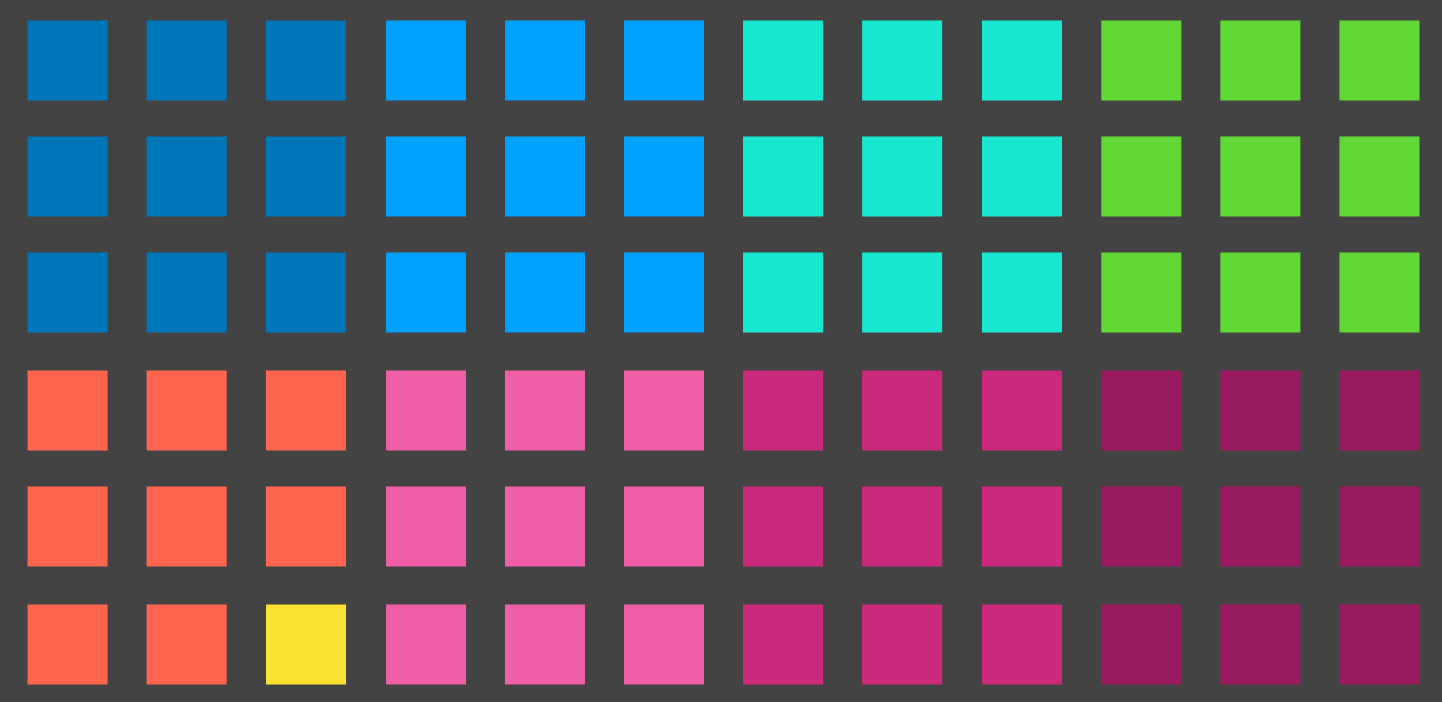


Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

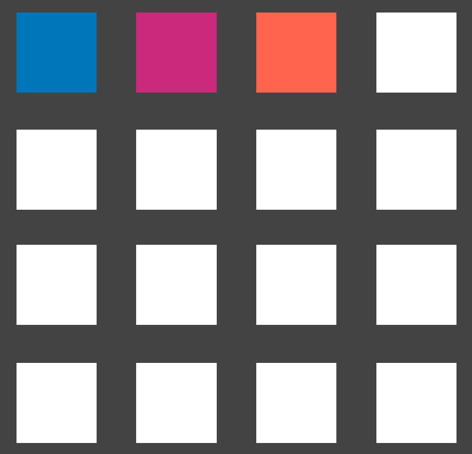


n total pages, **divided into blocks**

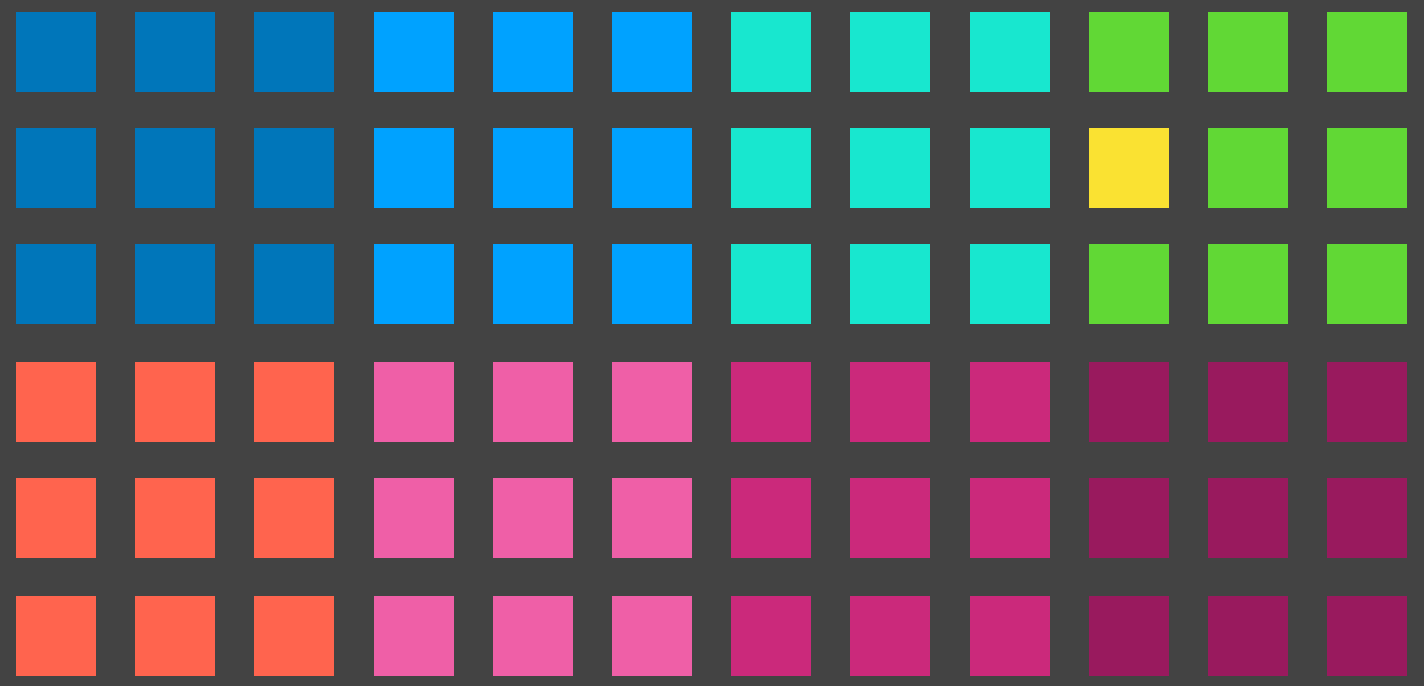


Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

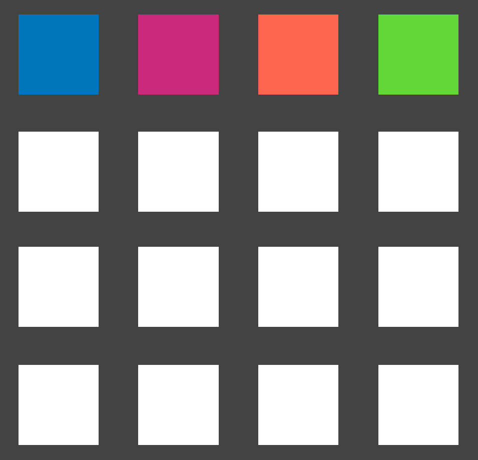


n total pages, **divided into blocks**

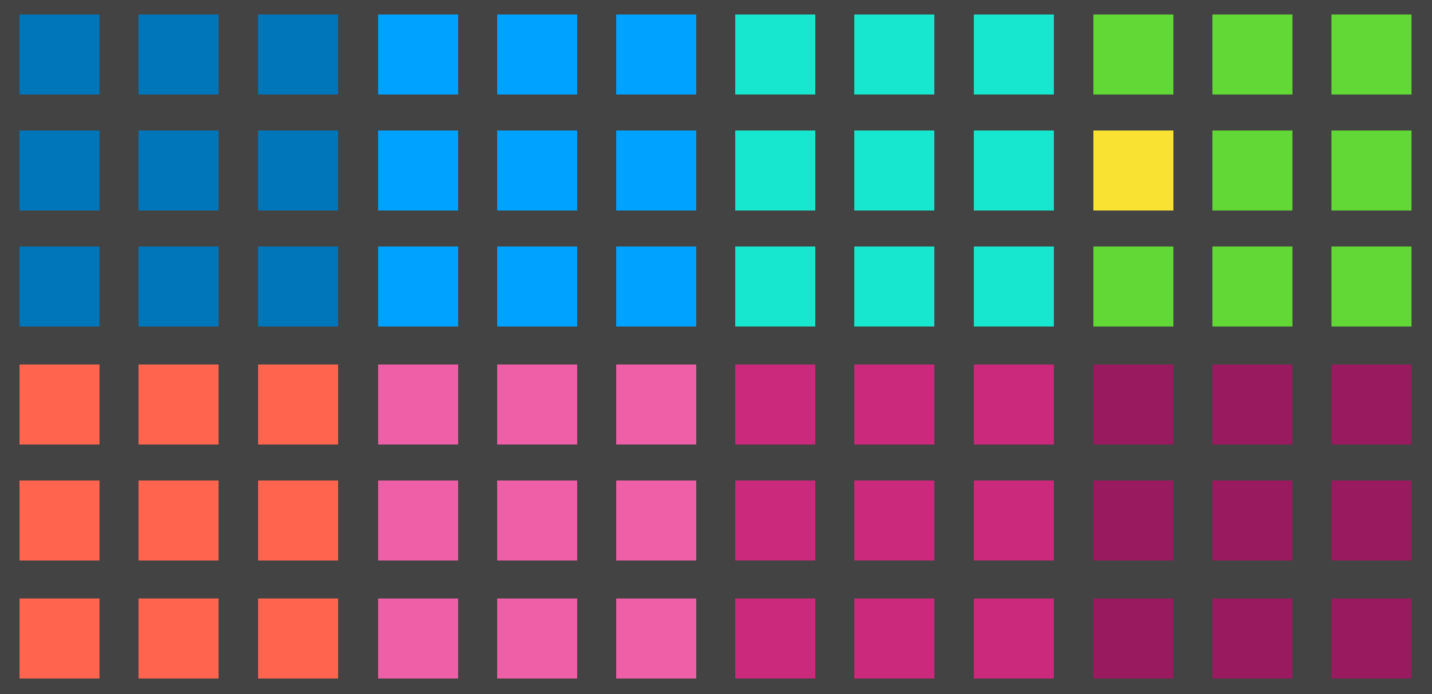


Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k



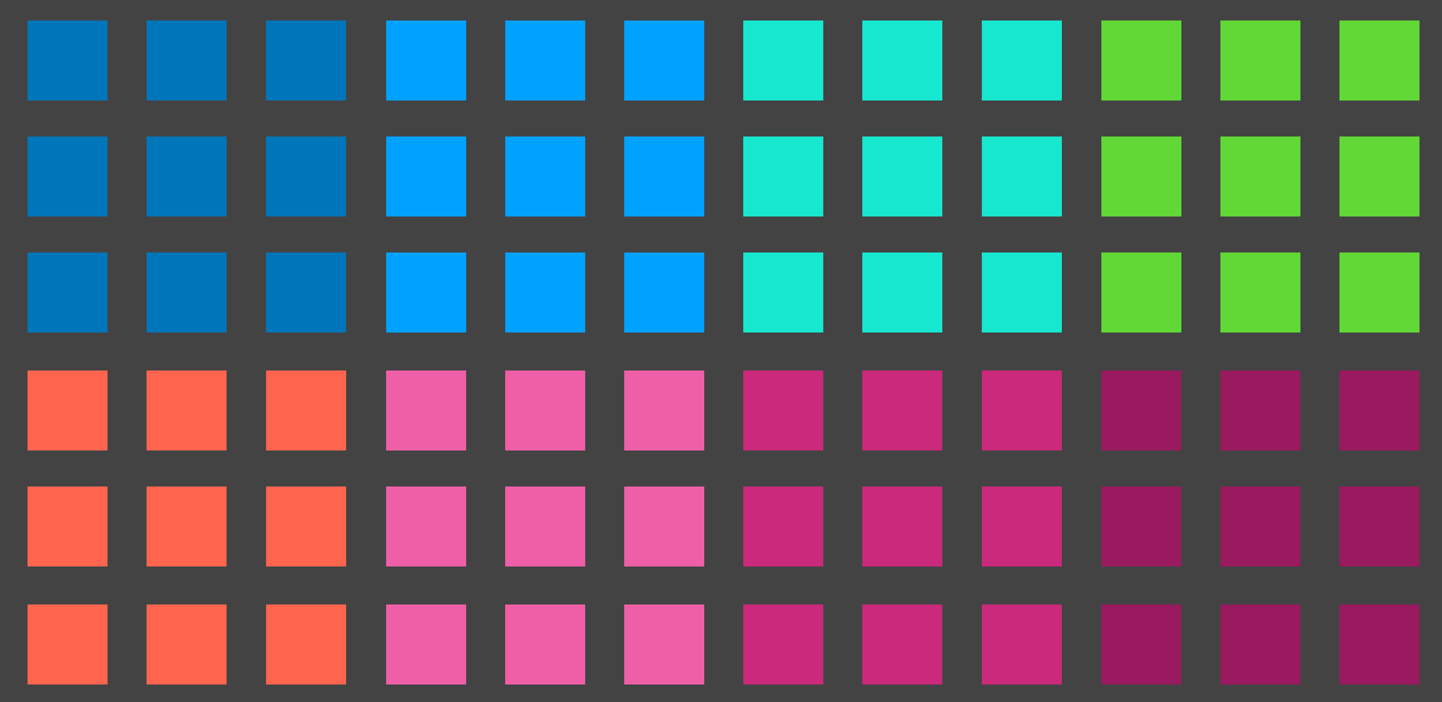
n total pages, **divided into blocks**



Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

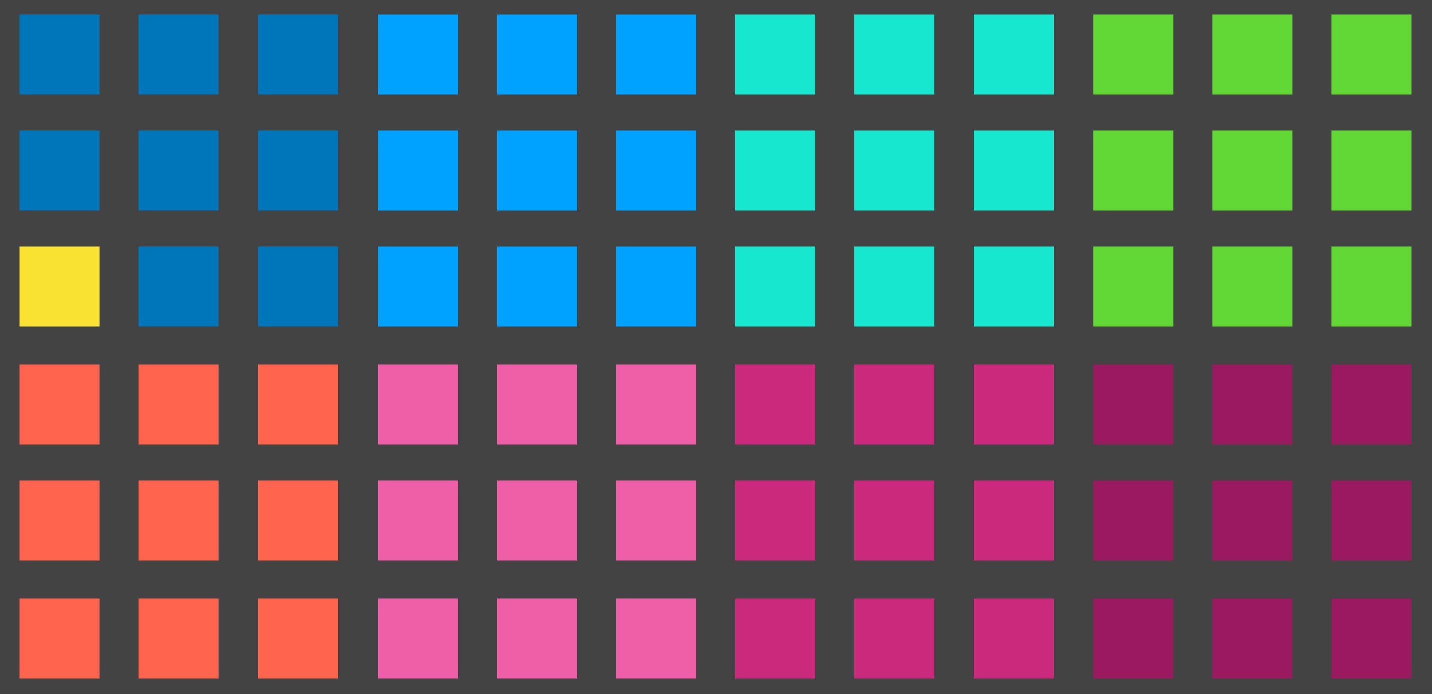
n total pages, **divided into blocks**



Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

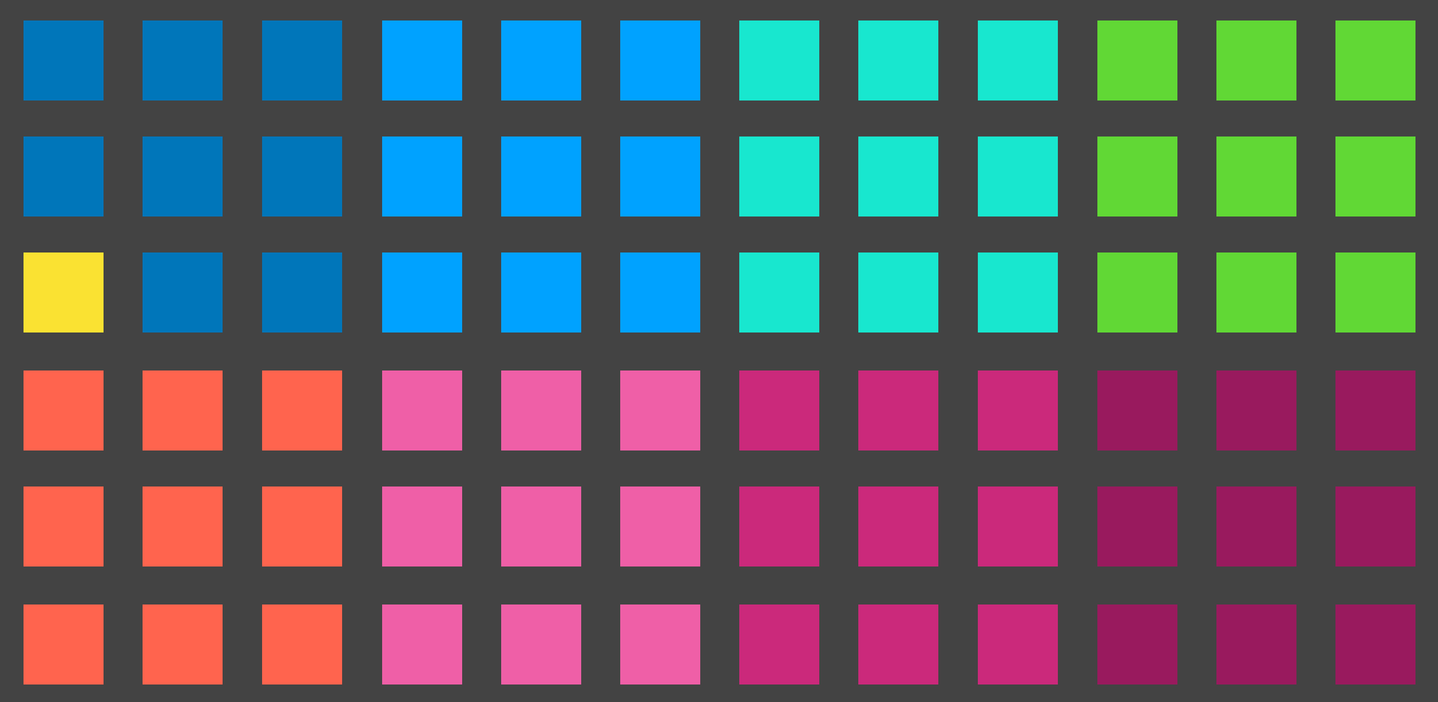
n total pages, **divided into blocks**



Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

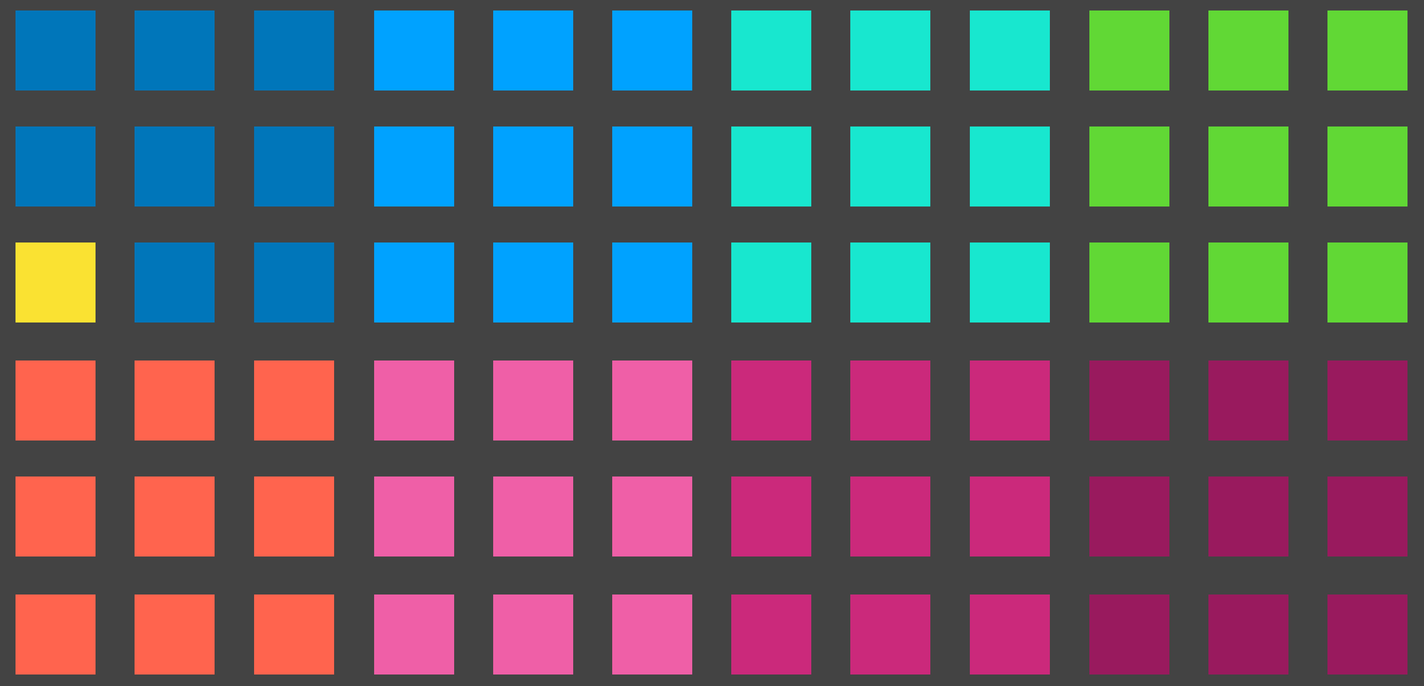
n total pages, **divided into blocks**



Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

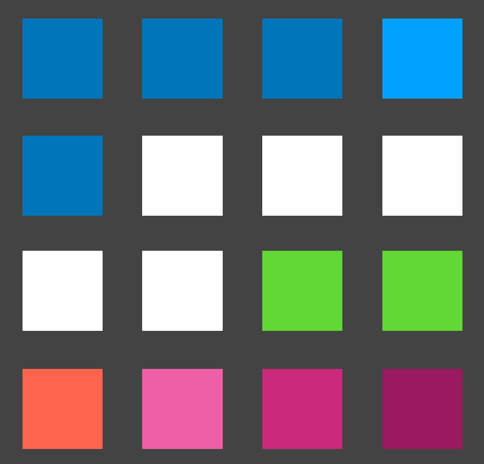
Cache of size k

n total pages, **divided into blocks**

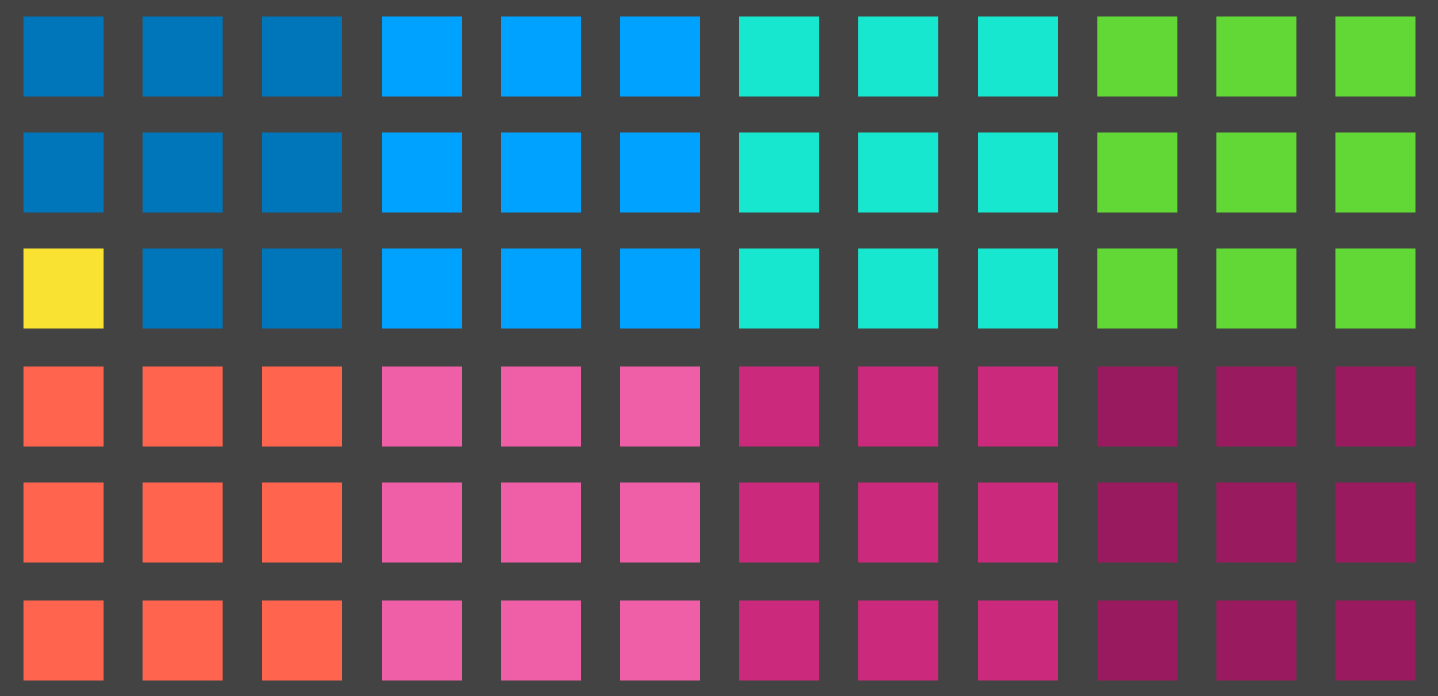


Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

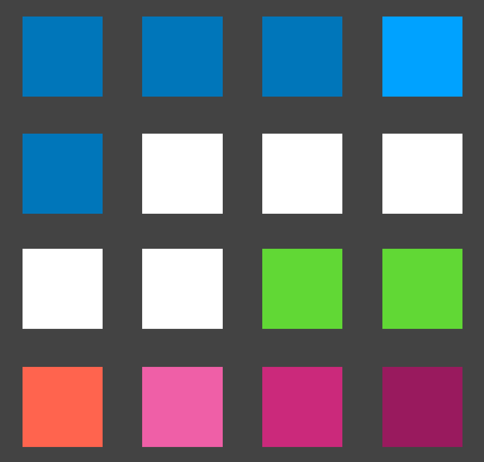


n total pages, **divided into blocks**

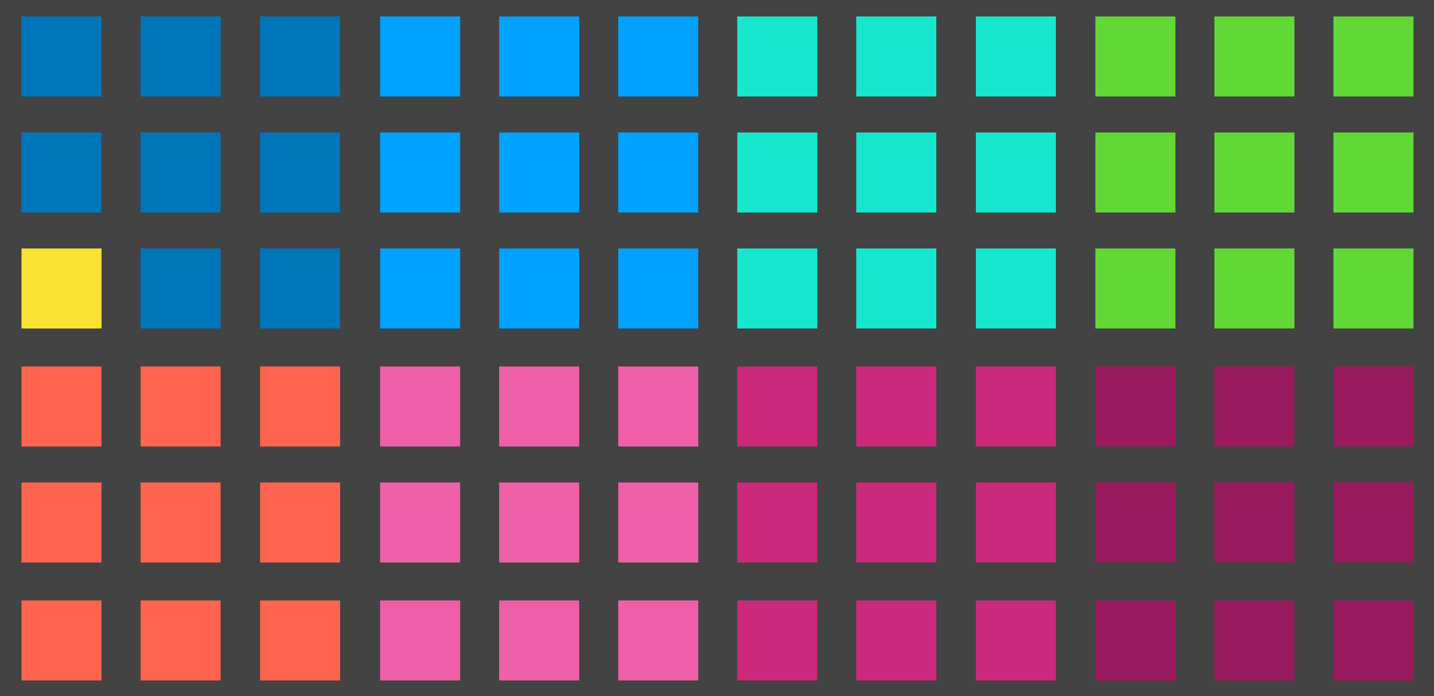


Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k



n total pages, **divided into blocks**

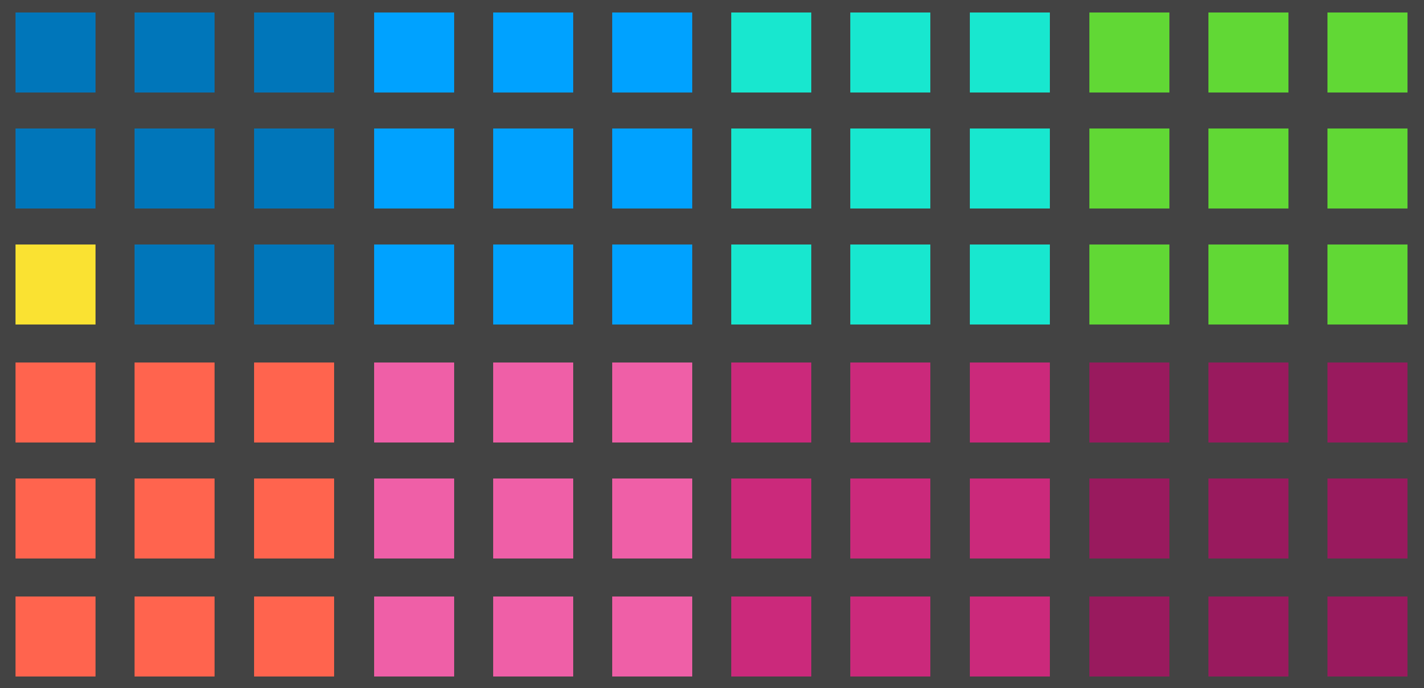


Goal is to minimize number of **blocks** fetched/evicted!

Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

n total pages, **divided into blocks**



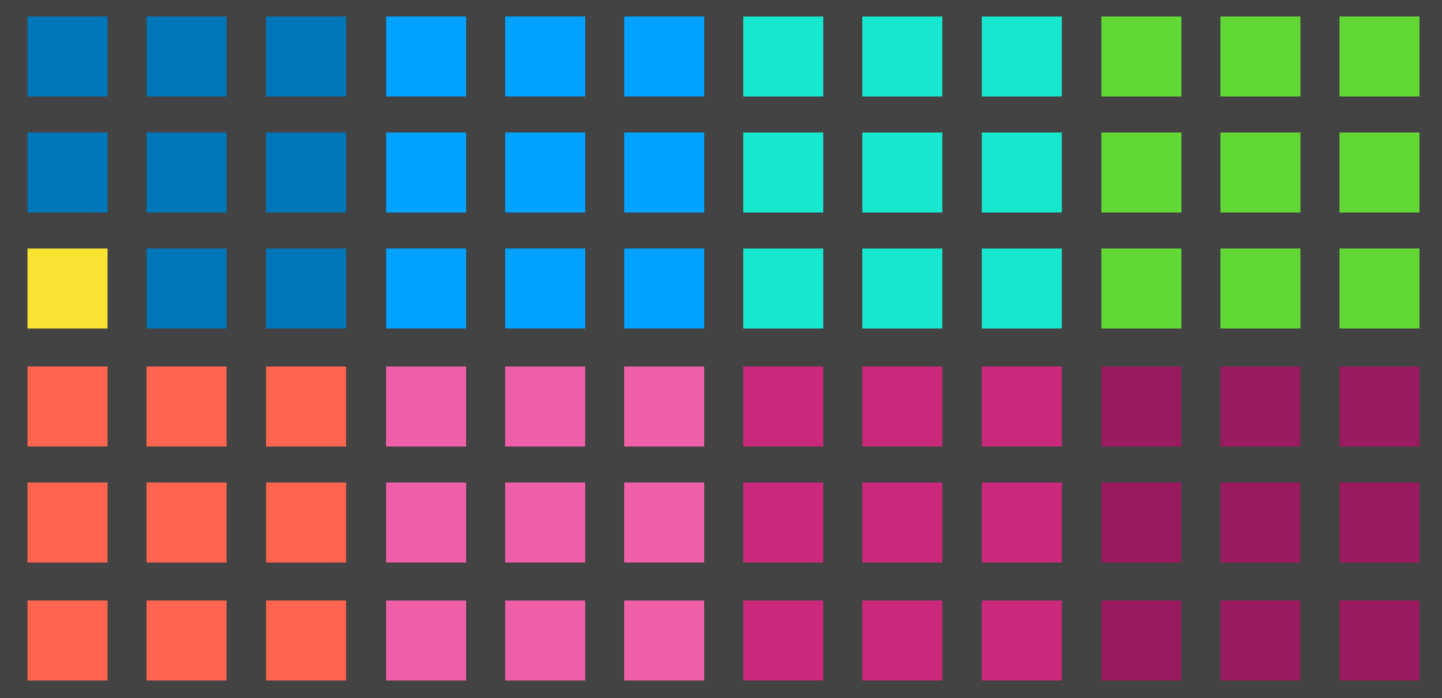
Goal is to minimize number of **blocks** fetched/evicted!

[Beckmann
Gibbons
McGuffey
SPAA 21]

Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

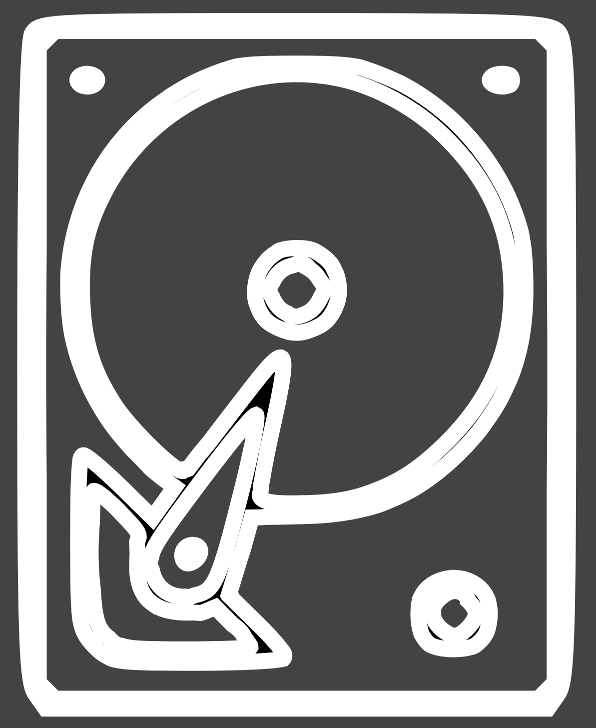
Cache of size k

n total pages, **divided into blocks**



Goal is to minimize number of **blocks** fetched/evicted!

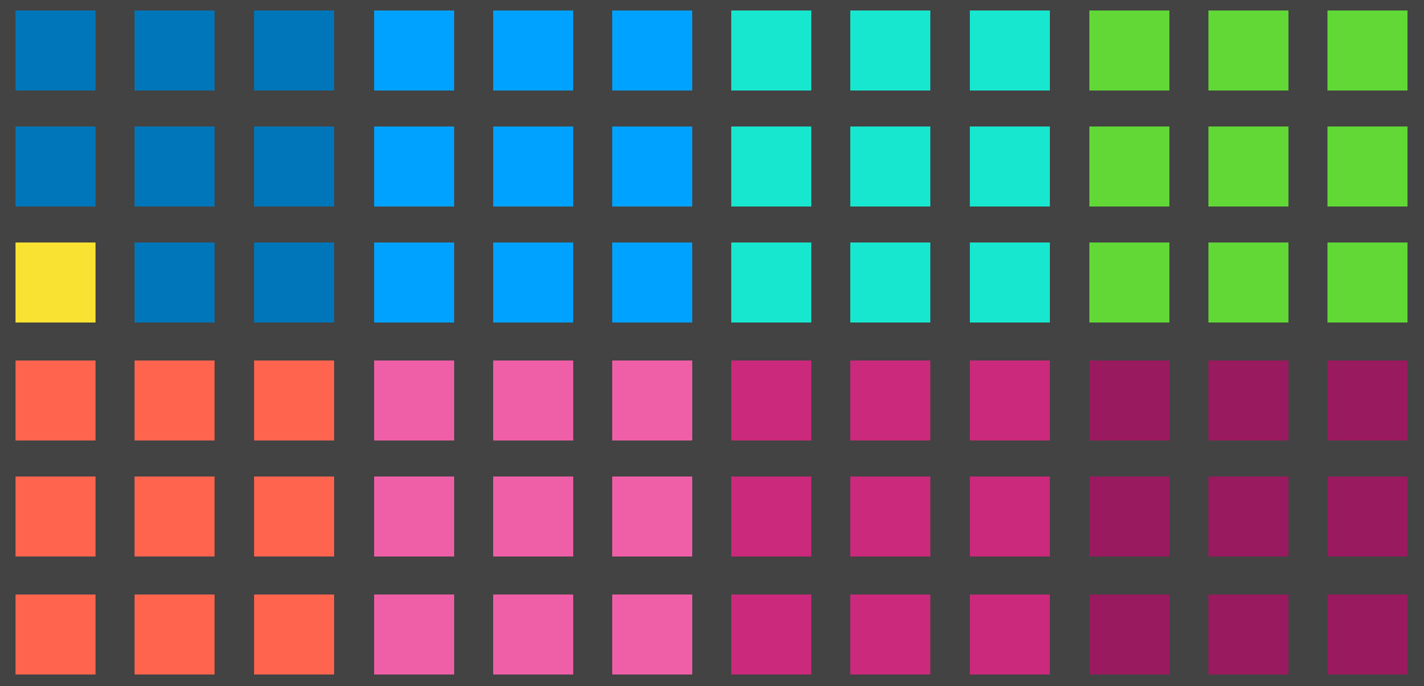
[Beckmann Gibbons McGuffey SPAA 21]



Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

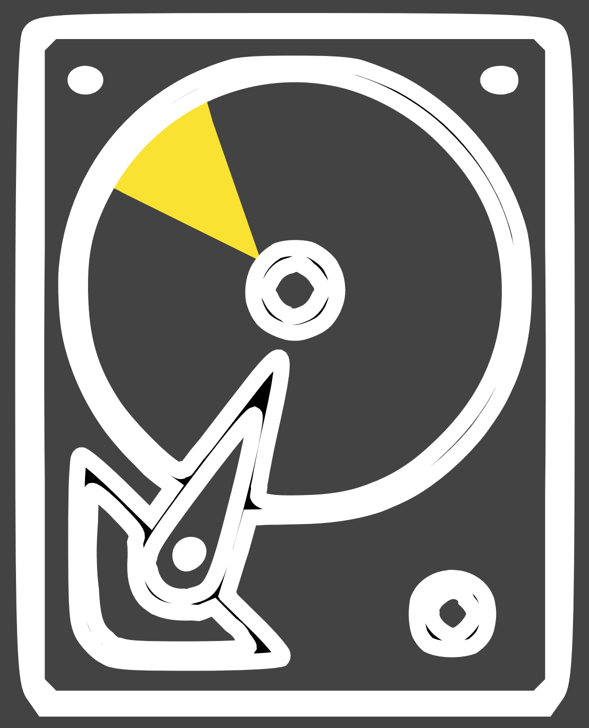
Cache of size k

n total pages, **divided into blocks**



Goal is to minimize number of **blocks** fetched/evicted!

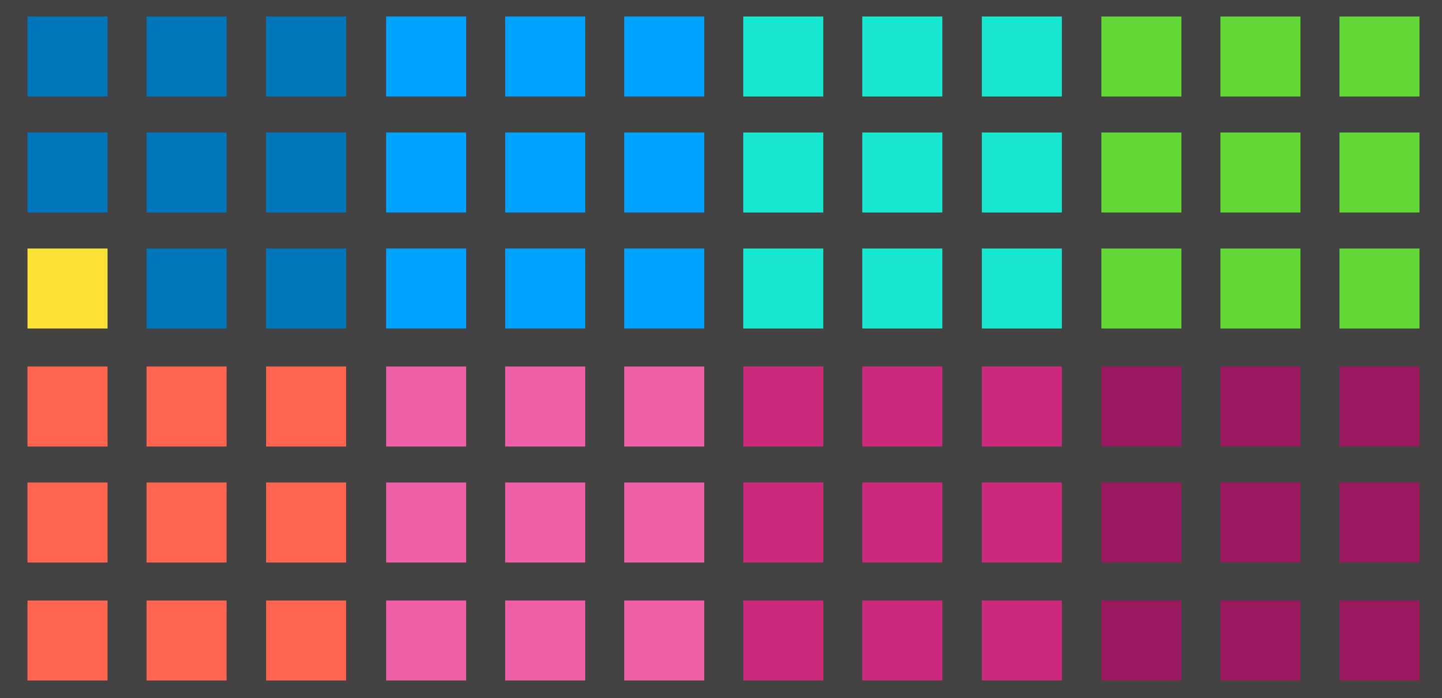
[Beckmann Gibbons McGuffey SPAA 21]



Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

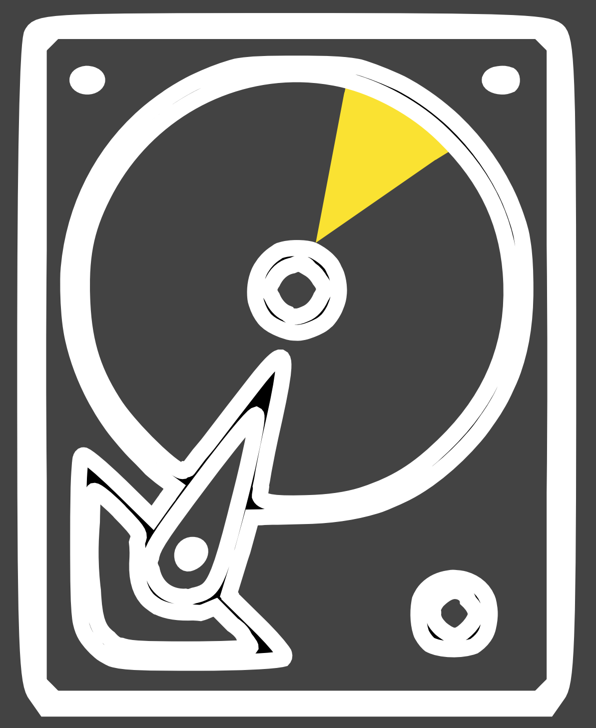
Cache of size k

n total pages, **divided into blocks**



Goal is to minimize number of **blocks** fetched/evicted!

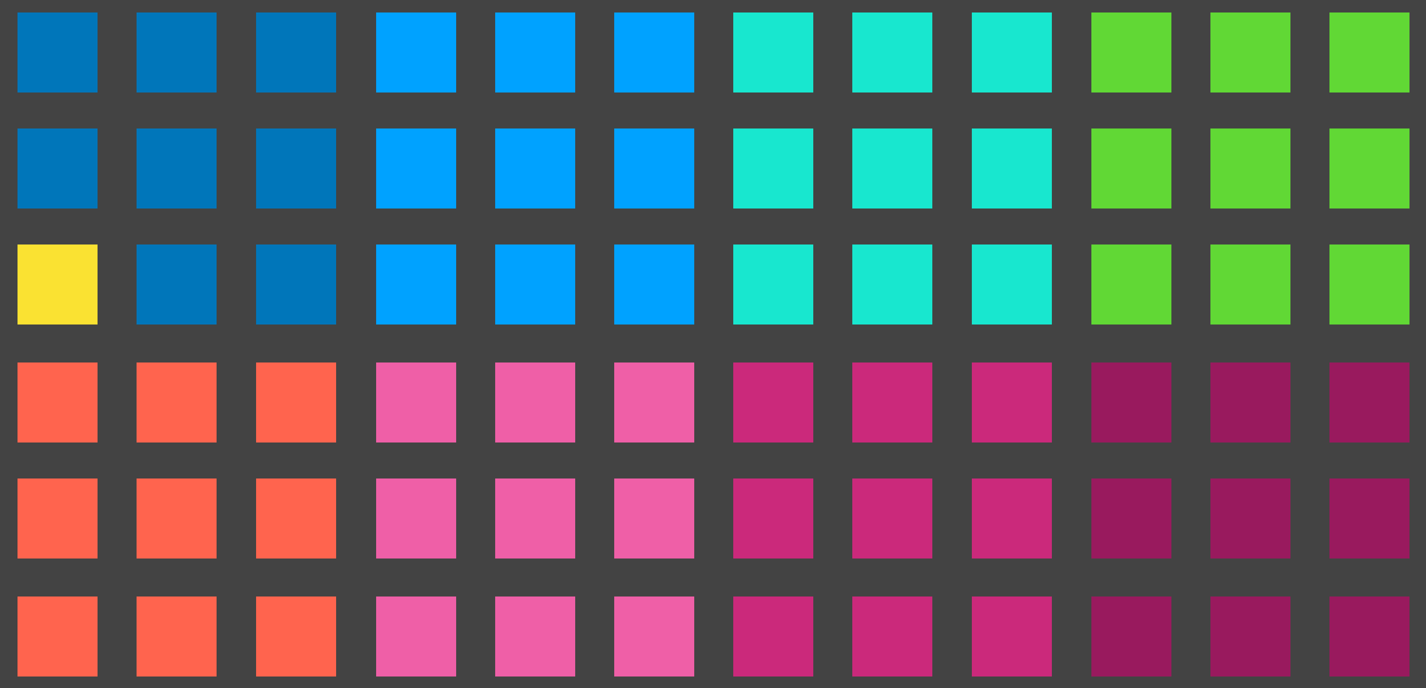
[Beckmann Gibbons McGuffey SPAA 21]



Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

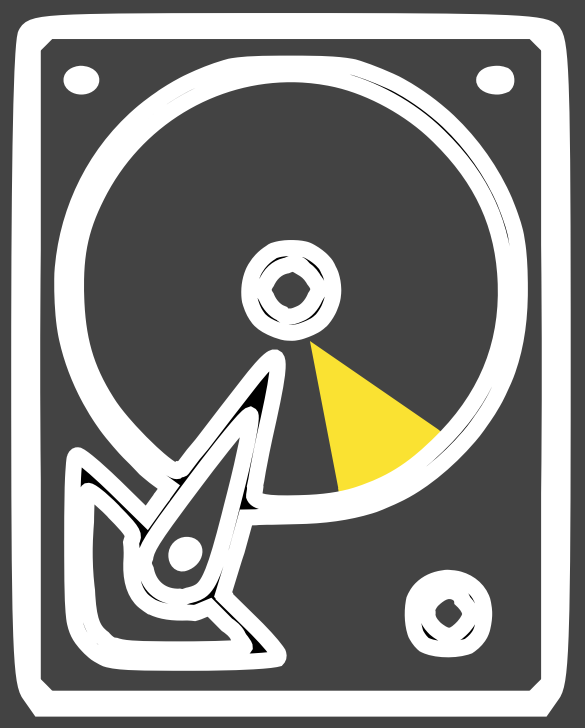
Cache of size k

n total pages, **divided into blocks**



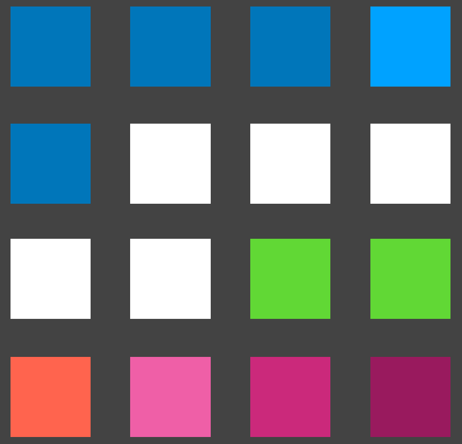
Goal is to minimize number of **blocks** fetched/evicted!

[Beckmann Gibbons McGuffey SPAA 21]

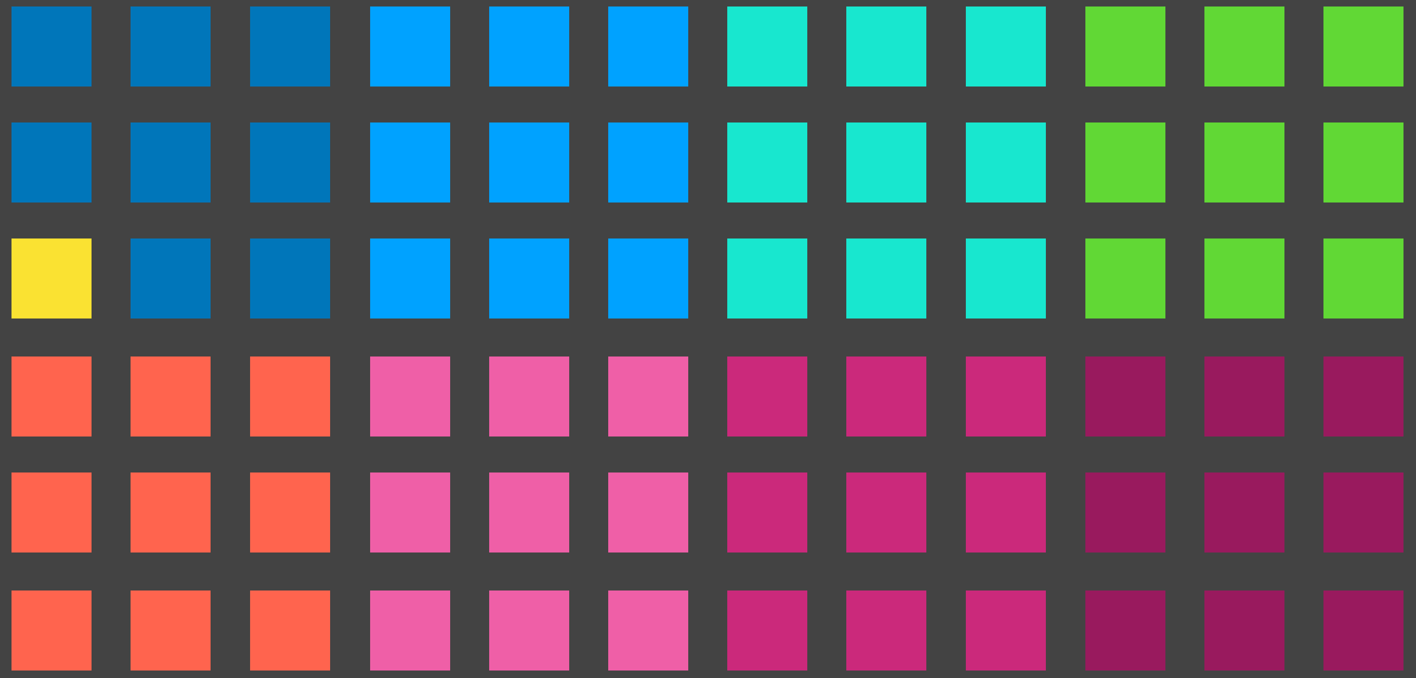


Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

Cache of size k

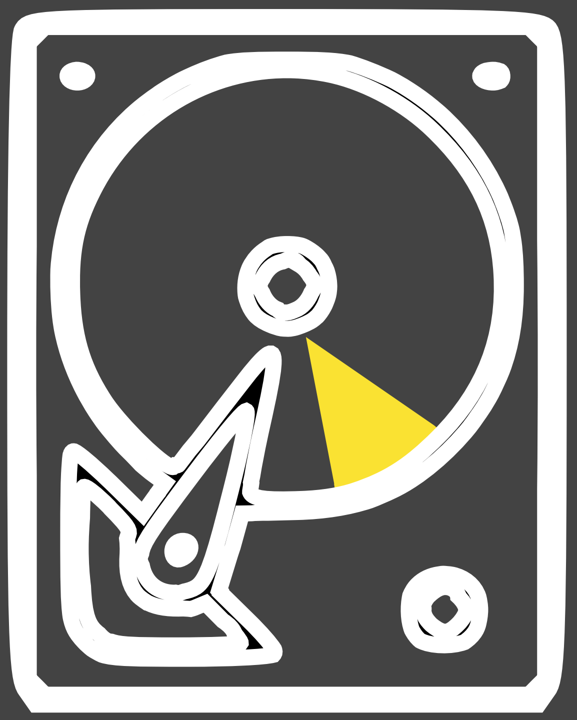


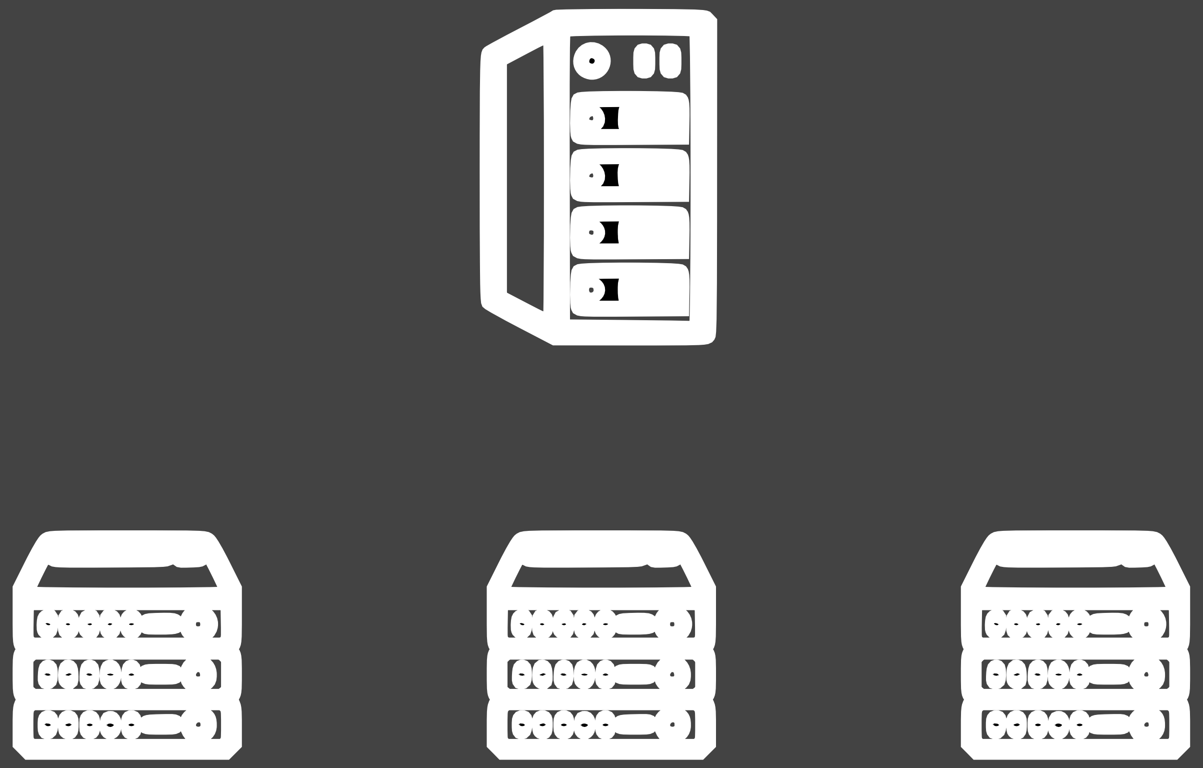
n total pages, **divided into blocks**



Goal is to minimize number of **blocks** fetched/evicted!

[Beckmann Gibbons McGuffey SPAA 21]

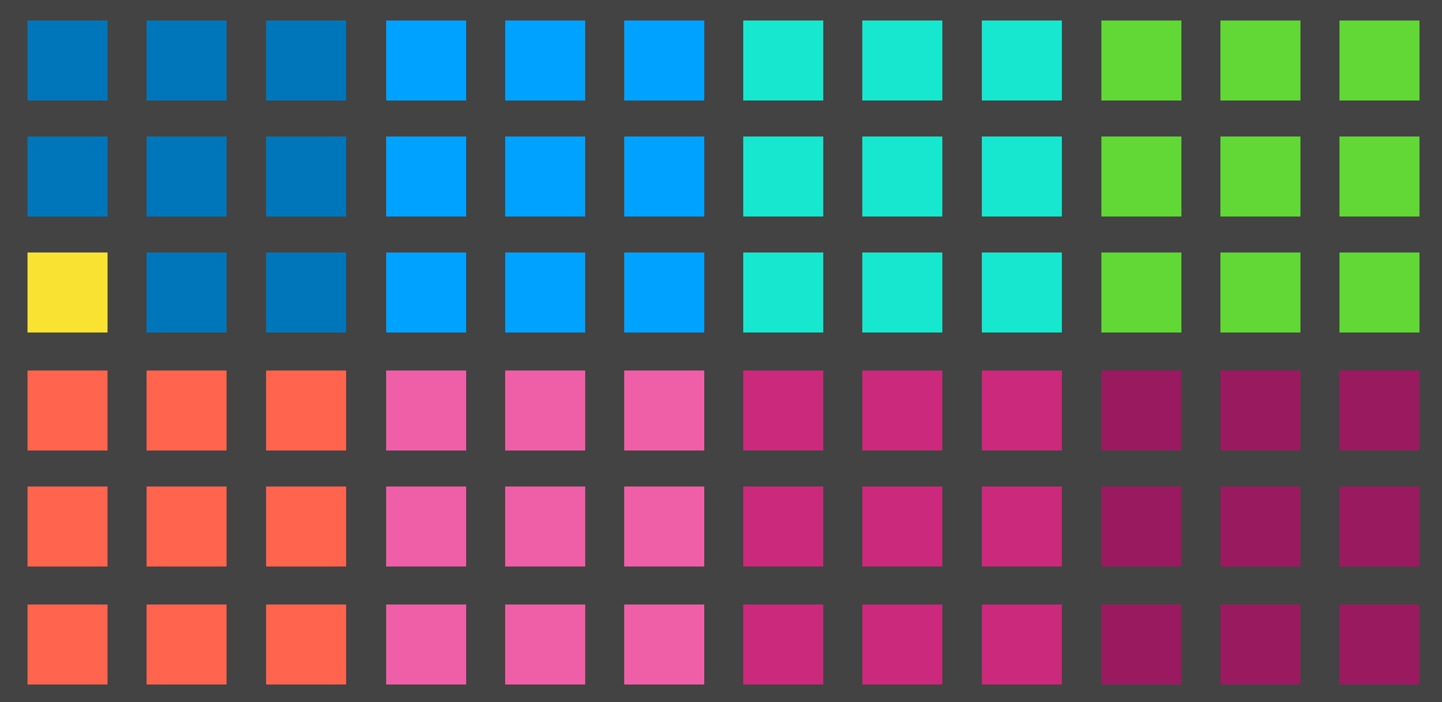




Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

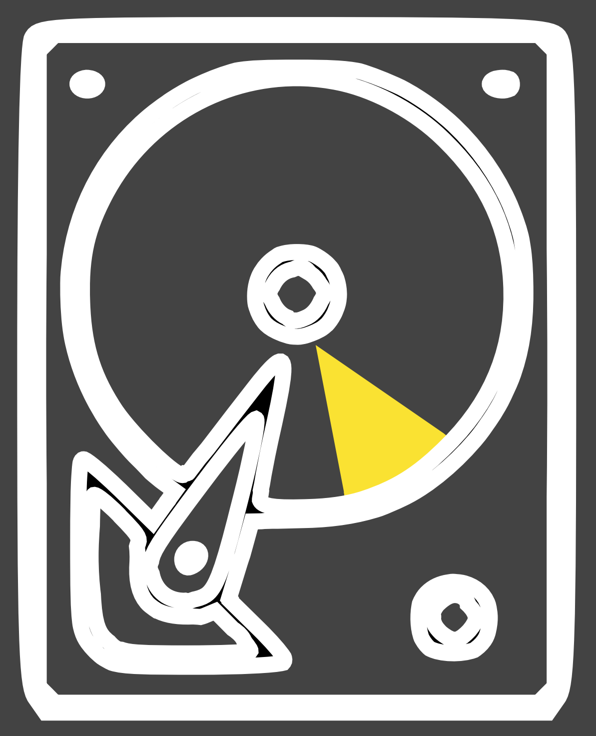
Cache of size k

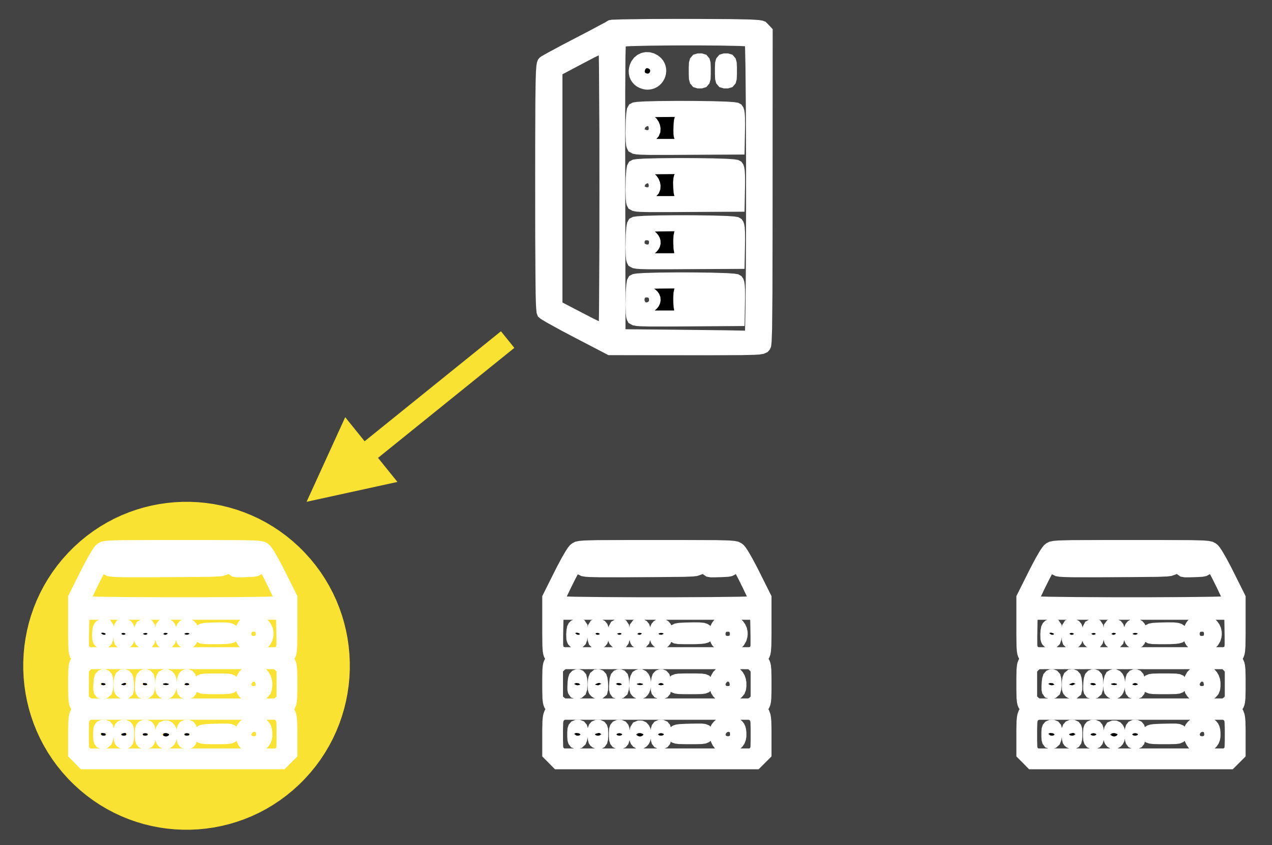
n total pages, **divided into blocks**



Goal is to minimize number of **blocks** fetched/evicted!

[Beckmann Gibbons McGuffey SPAA 21]

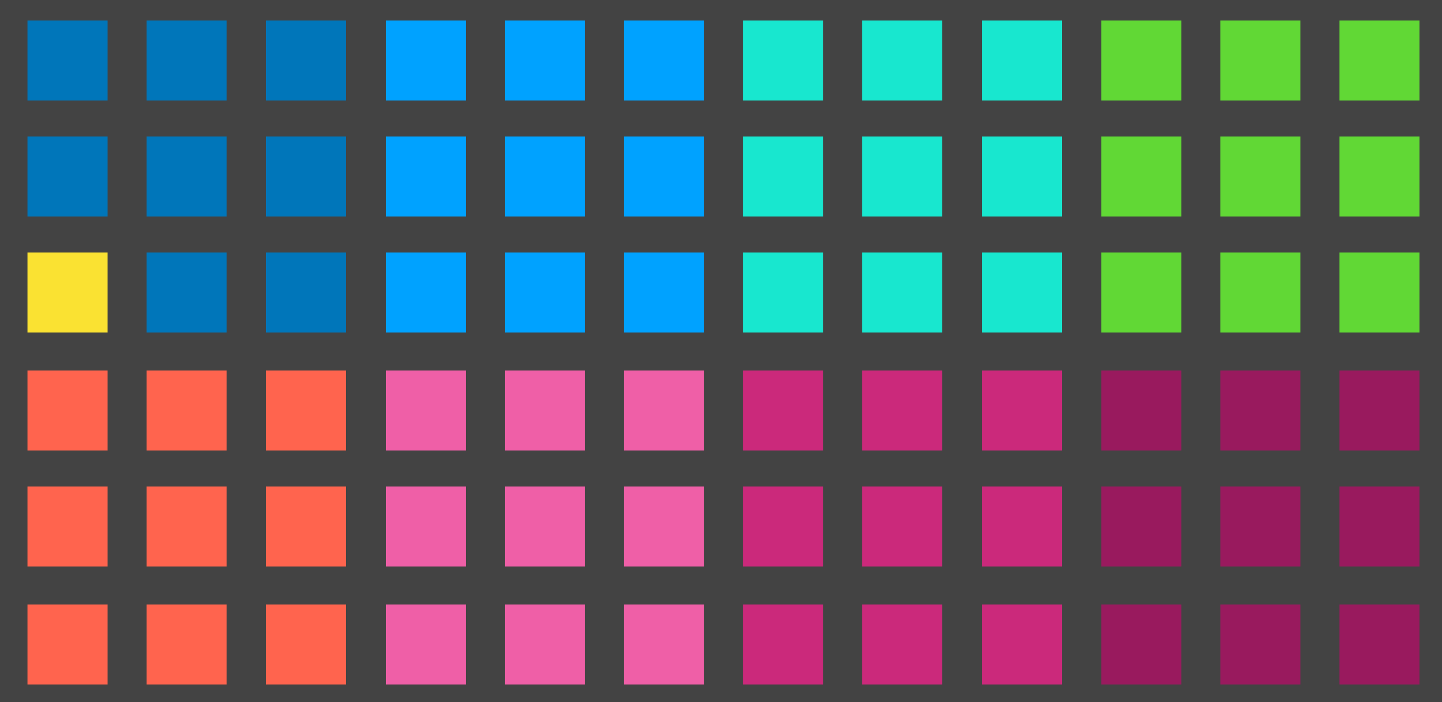




Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

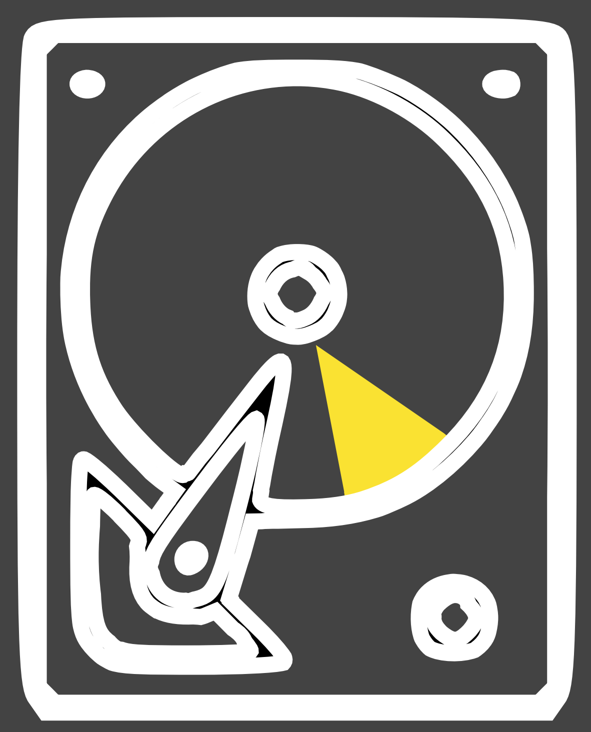
Cache of size k

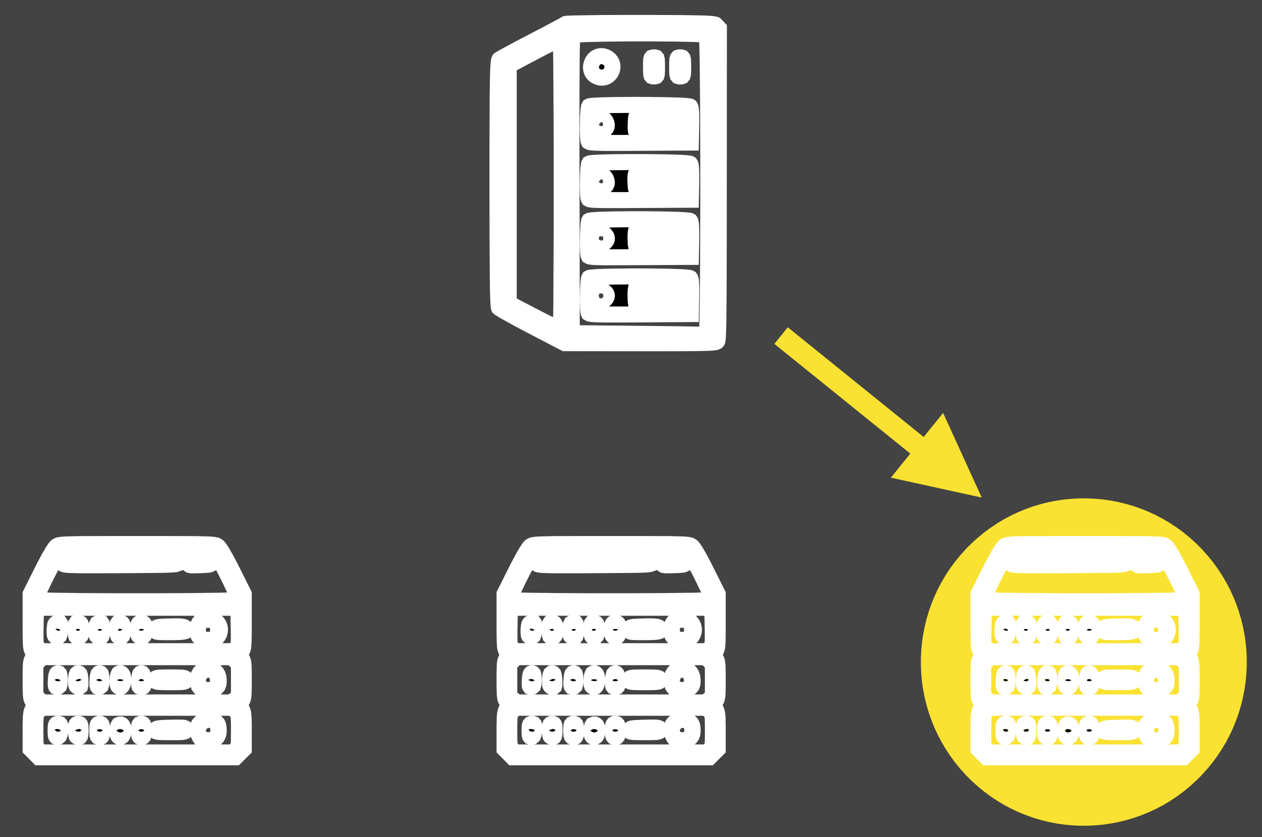
n total pages, **divided into blocks**



Goal is to minimize number of **blocks** fetched/evicted!

[Beckmann Gibbons McGuffey SPAA 21]

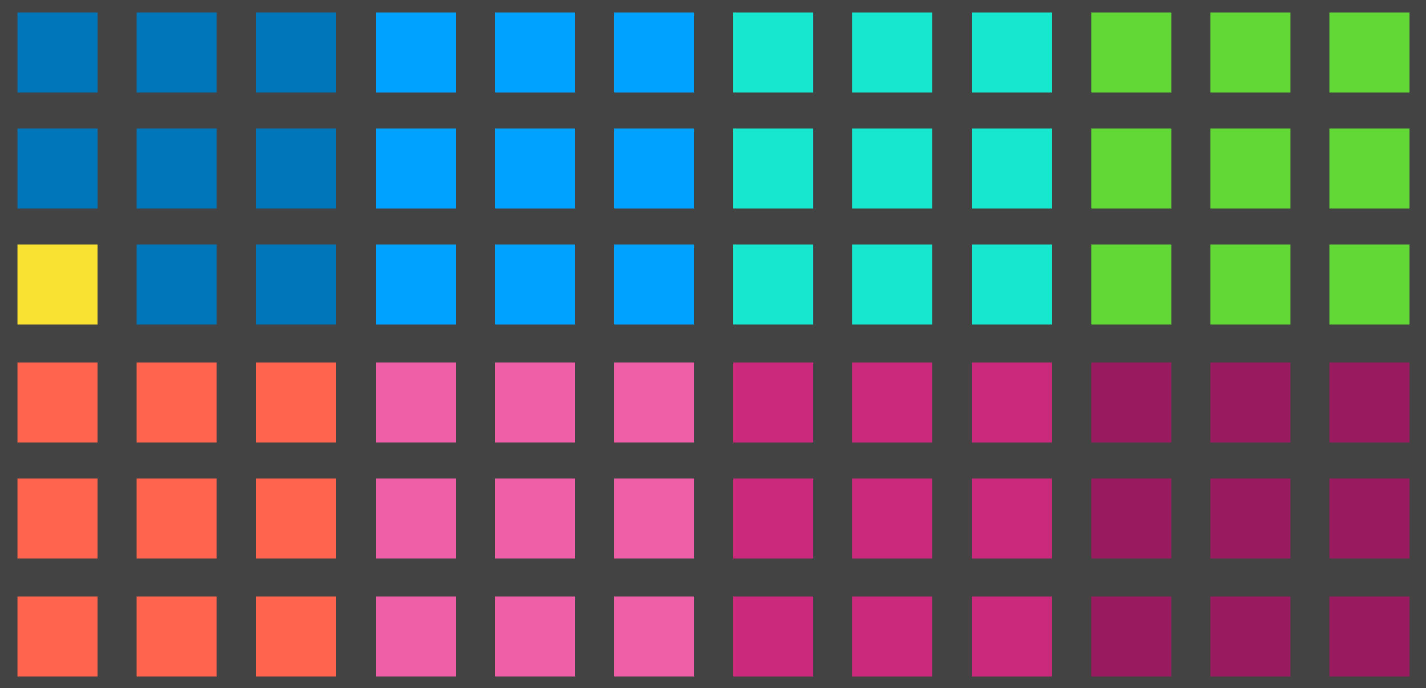




Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

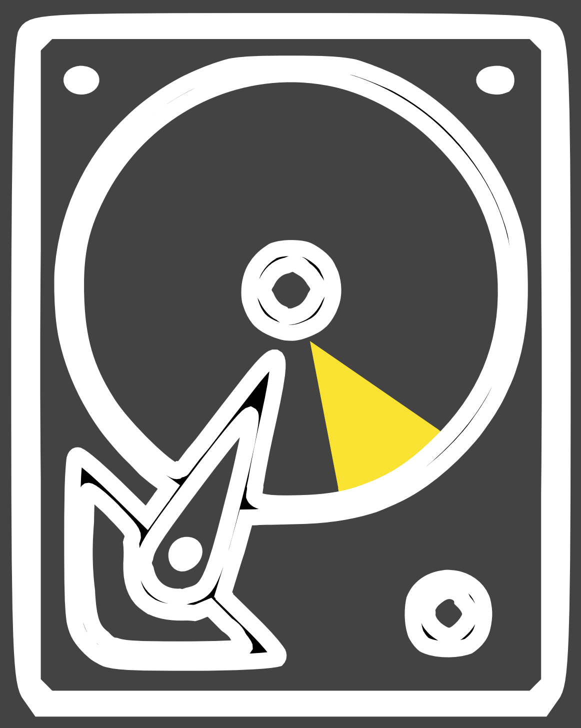
Cache of size k

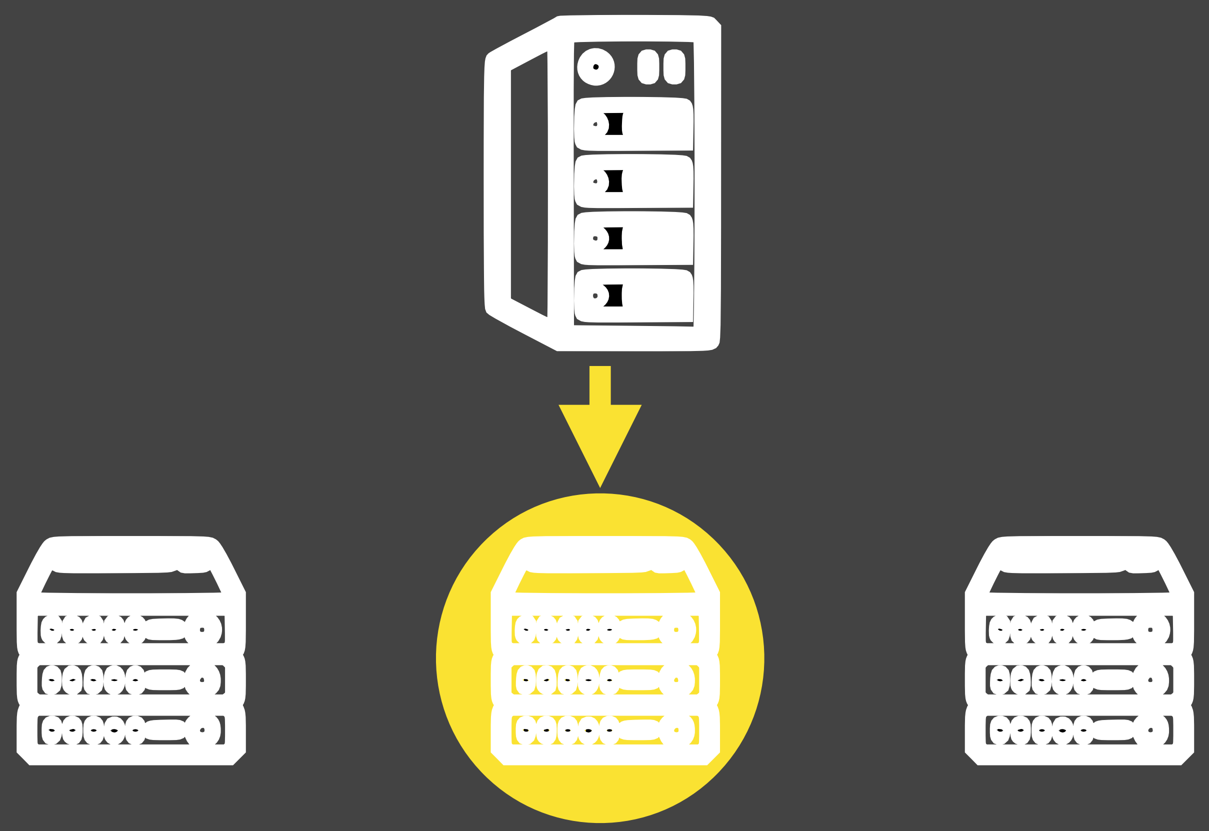
n total pages, **divided into blocks**



Goal is to minimize number of **blocks** fetched/evicted!

[Beckmann Gibbons McGuffey SPAA 21]

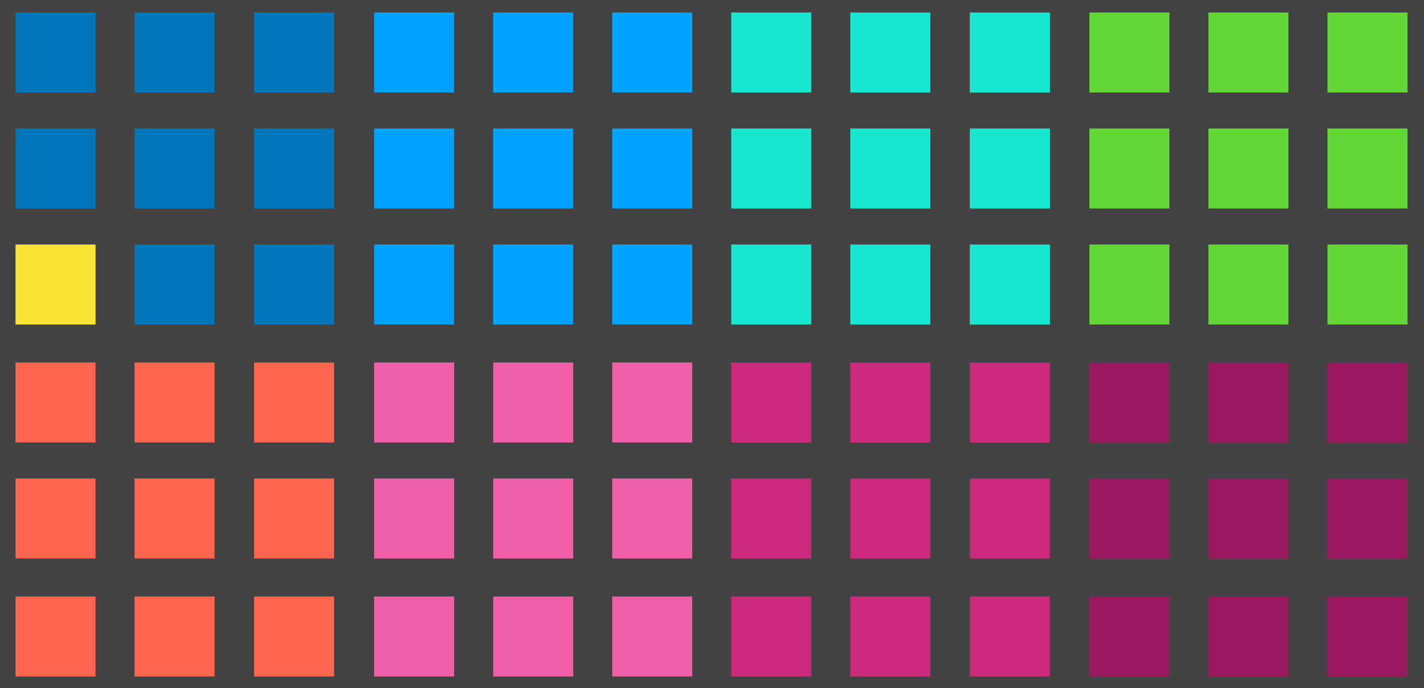




Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

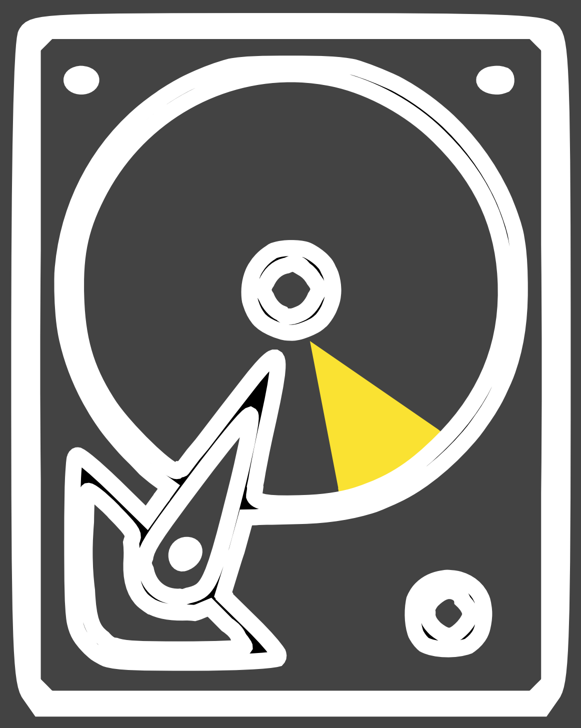
Cache of size k

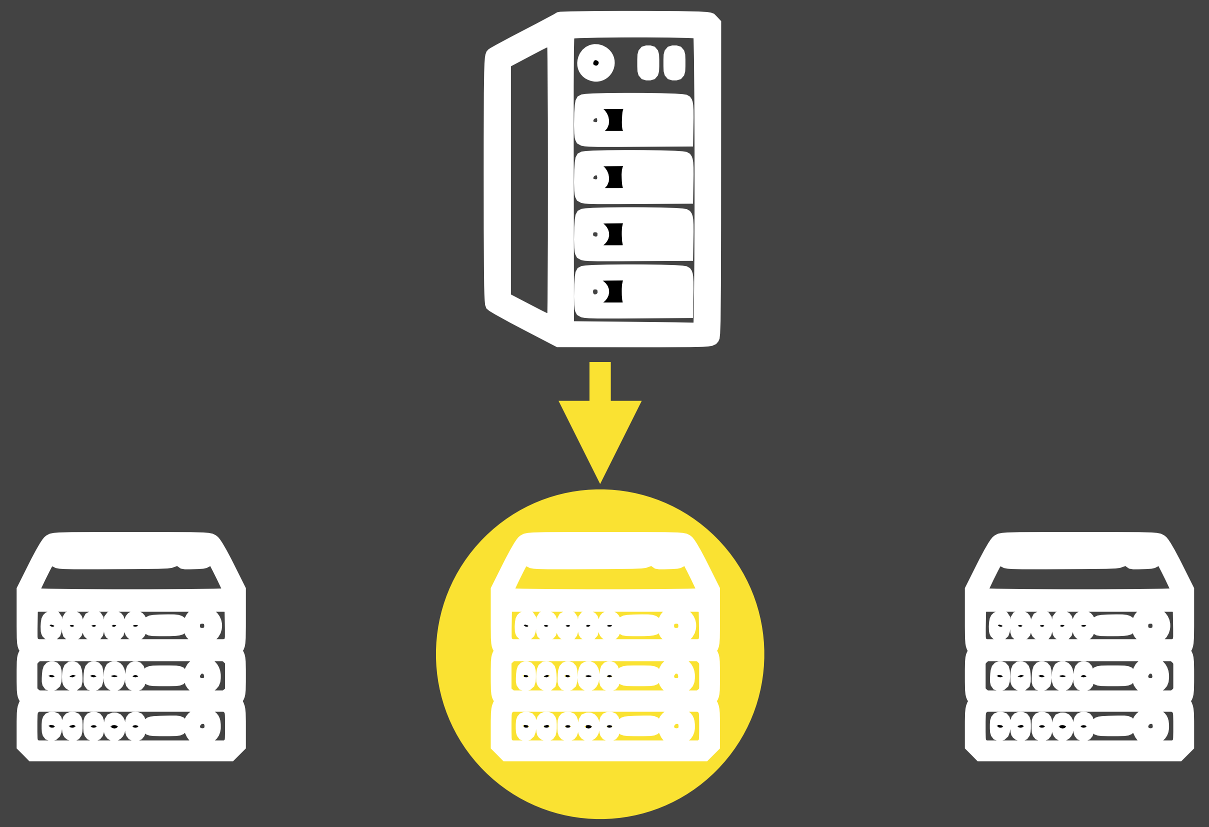
n total pages, **divided into blocks**



Goal is to minimize number of **blocks** fetched/evicted!

[Beckmann Gibbons McGuffey SPAA 21]



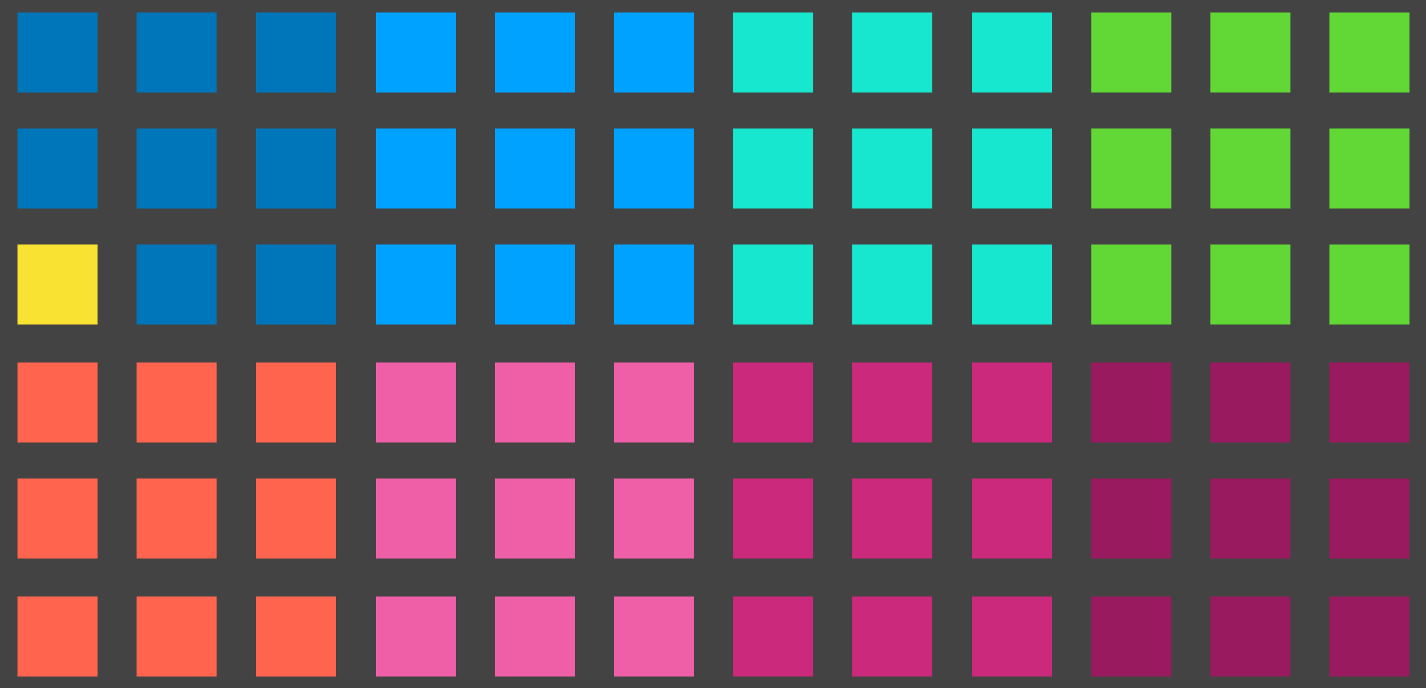


We give near-optimal algos using [GL. 20]!

Block-Aware Caching [Coester, Naor, L., Talmon SPAA 22]

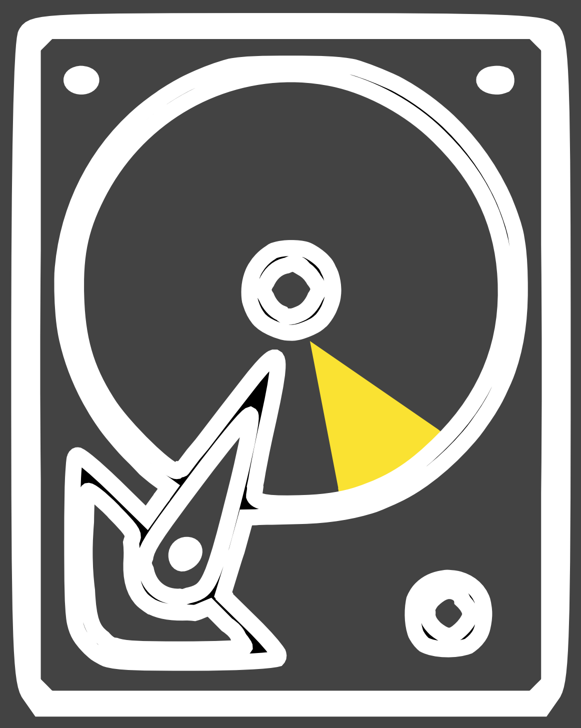
Cache of size k

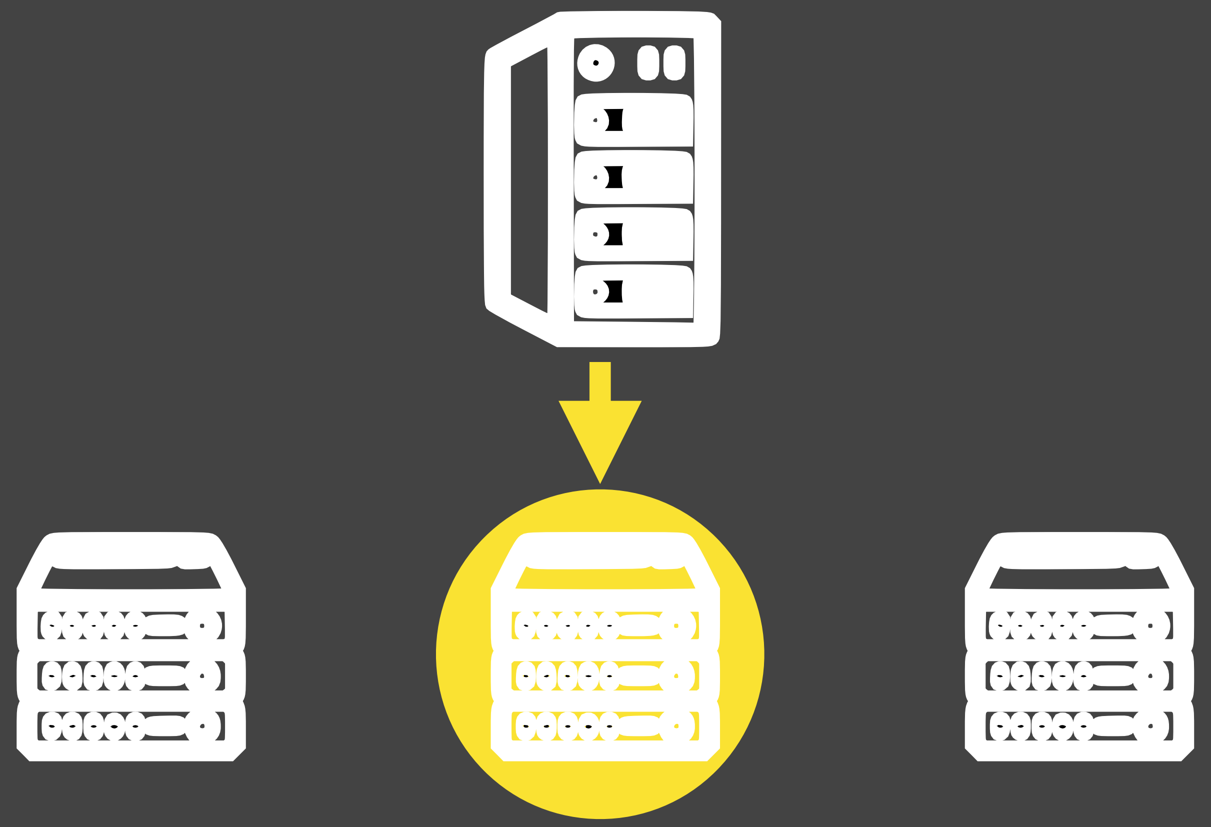
n total pages, **divided into blocks**



Goal is to minimize number of **blocks** fetched/evicted!

[Beckmann Gibbons McGuffey SPAA 21]





We give near-optimal algos using [GL. 20]!

Reduction to Online submodular cover!

Take Away I

[Gupta L. SODA 20]
[Coester, Naor, L., Talmon SPAA 22]

<u>**Q**</u>: What general classes of optimization problems can we solve online?

Take Away I

[Gupta L. SODA 20]
[Coester, Naor, L., Talmon SPAA 22]

Q: What general classes of optimization problems can we solve online?

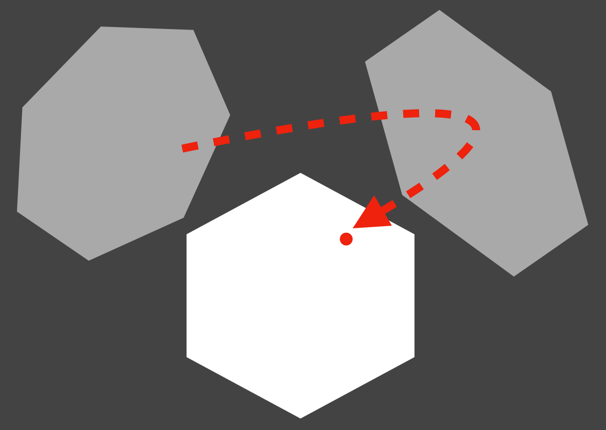
A: Any problem expressible as Submodular Cover!

Outline

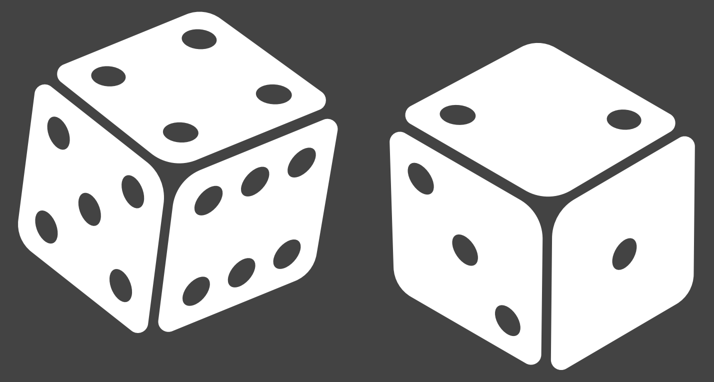
Theme I — Submodular Optimization

f(🍕 | 🥕) ≥ f(🍕 | 🥕, 🍩)

Theme II — Stable Algorithms



Theme III — Beyond Worst-Case Analysis



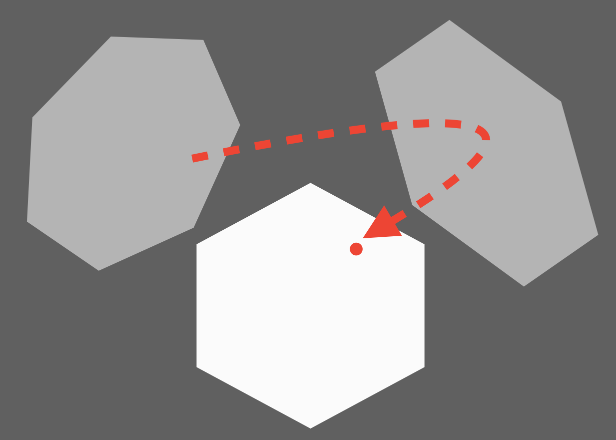
Conclusion

Outline

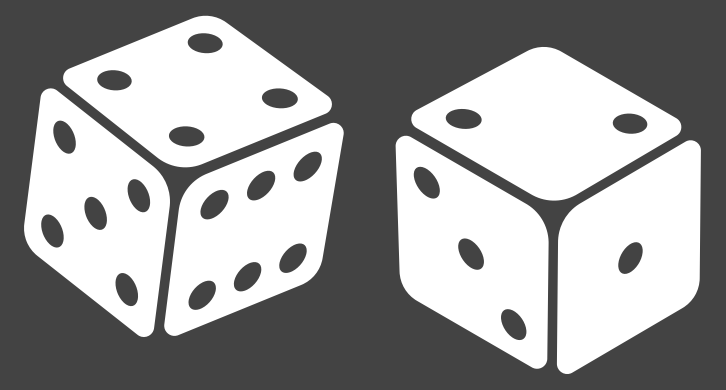
Theme I — Submodular Optimization

$f($🍕$\,|\,$🥕$) \geq f($🍕$\,|\,$🥕$,$🍩$)$

Theme II — Stable Algorithms



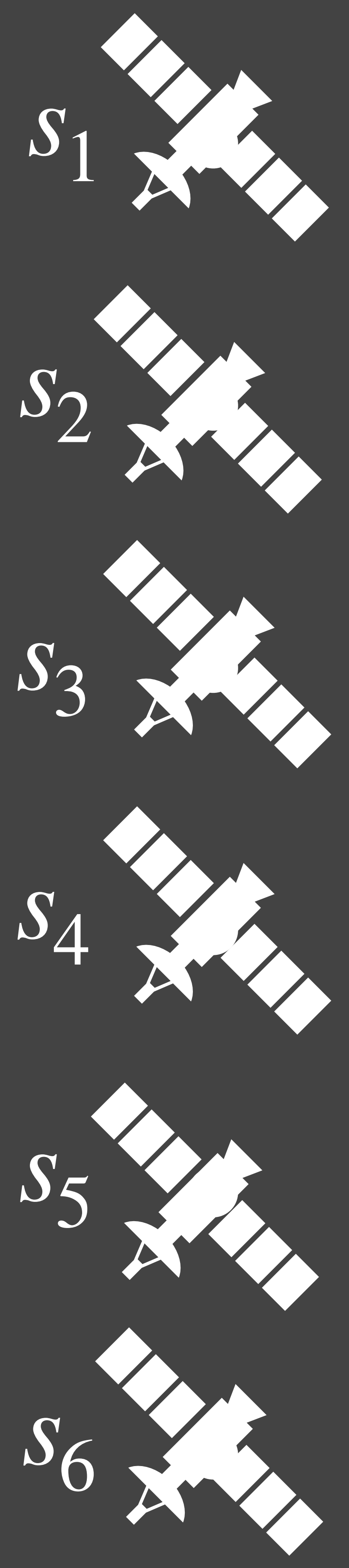
Theme III — Beyond Worst-Case Analysis



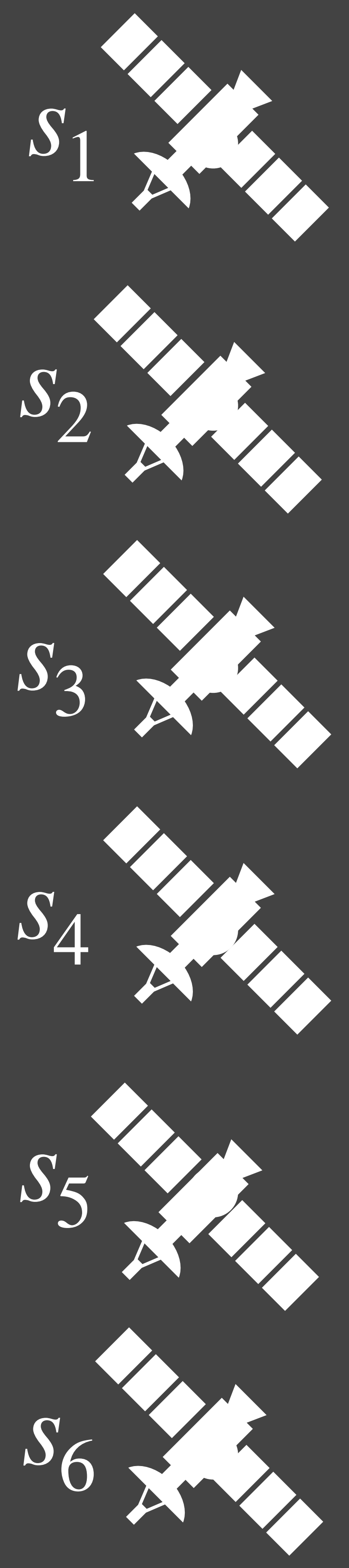
Conclusion

Theme II — Stable Algorithms

Moving to the Dynamic model

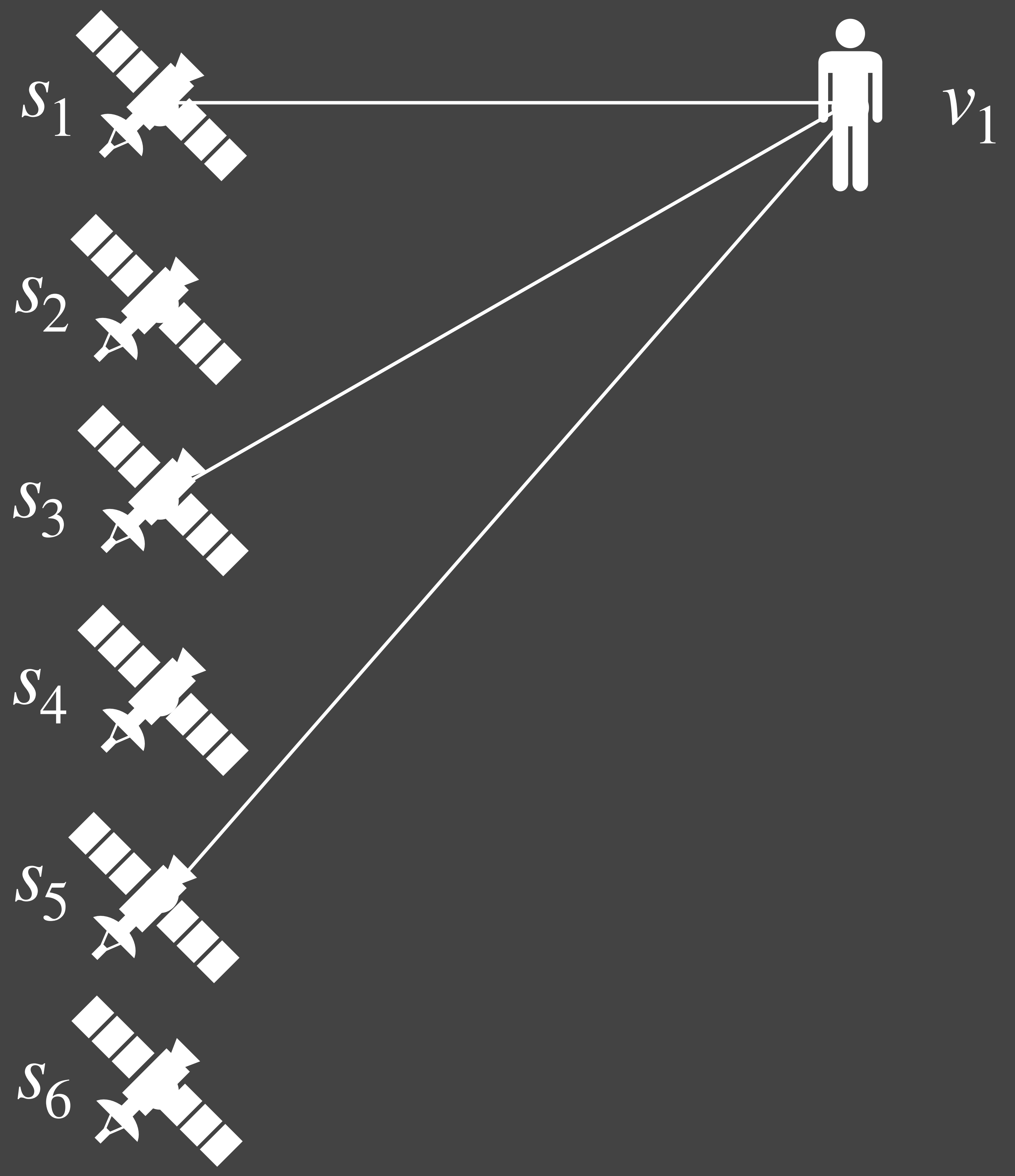


Moving to the Dynamic model



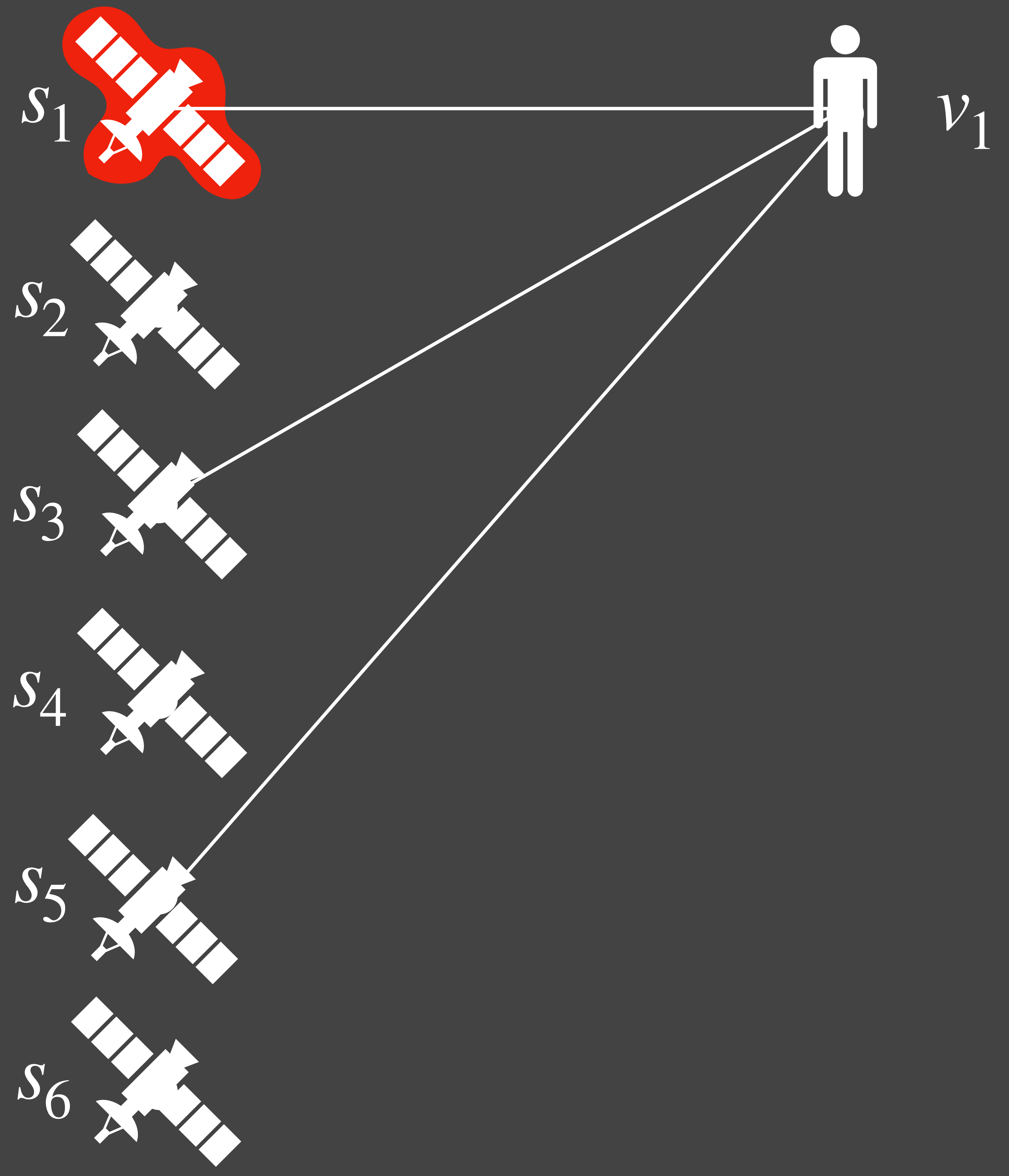
New model:
inserts AND deletes.

Moving to the Dynamic model



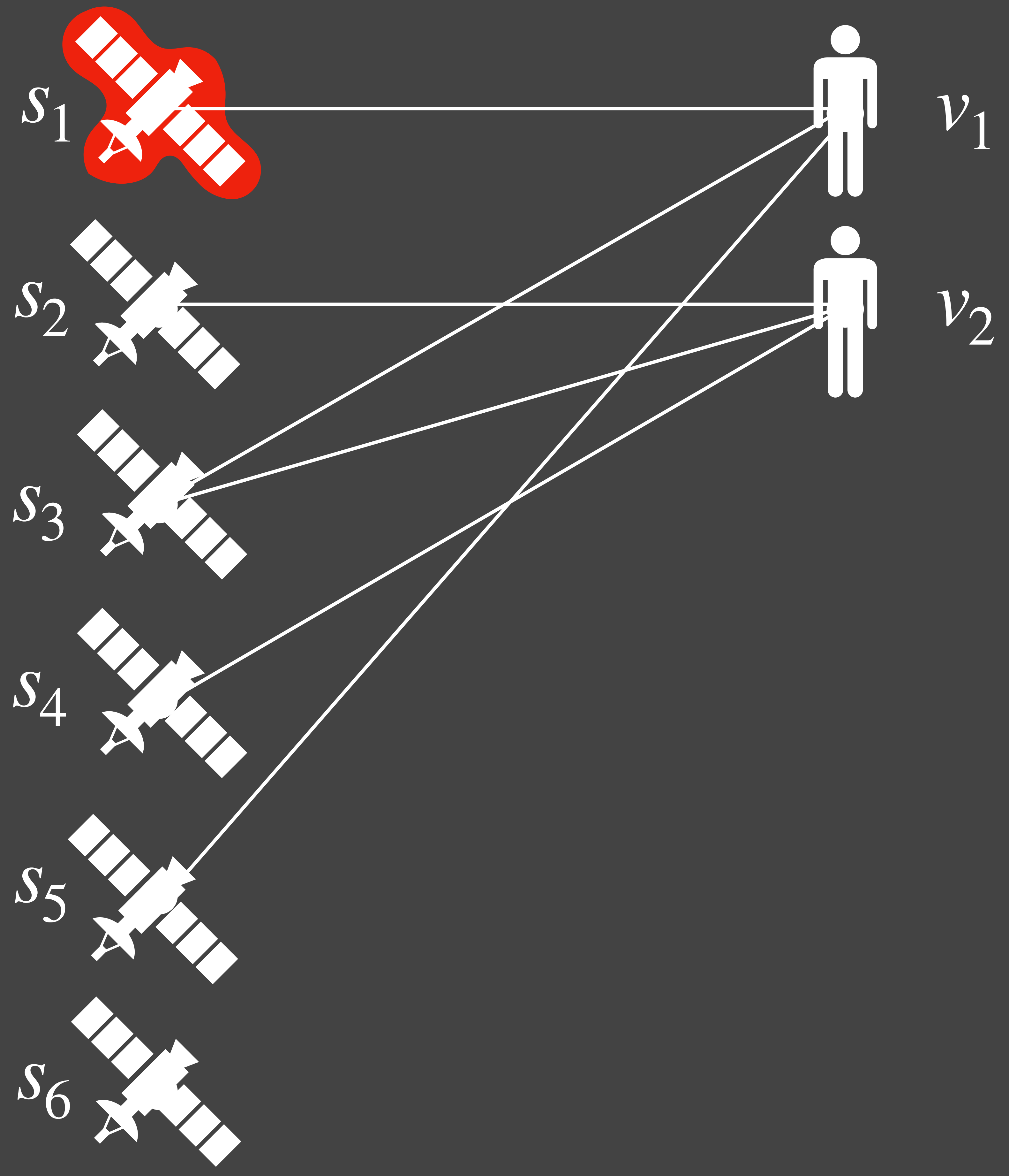
New model:
inserts AND deletes.

Moving to the Dynamic model



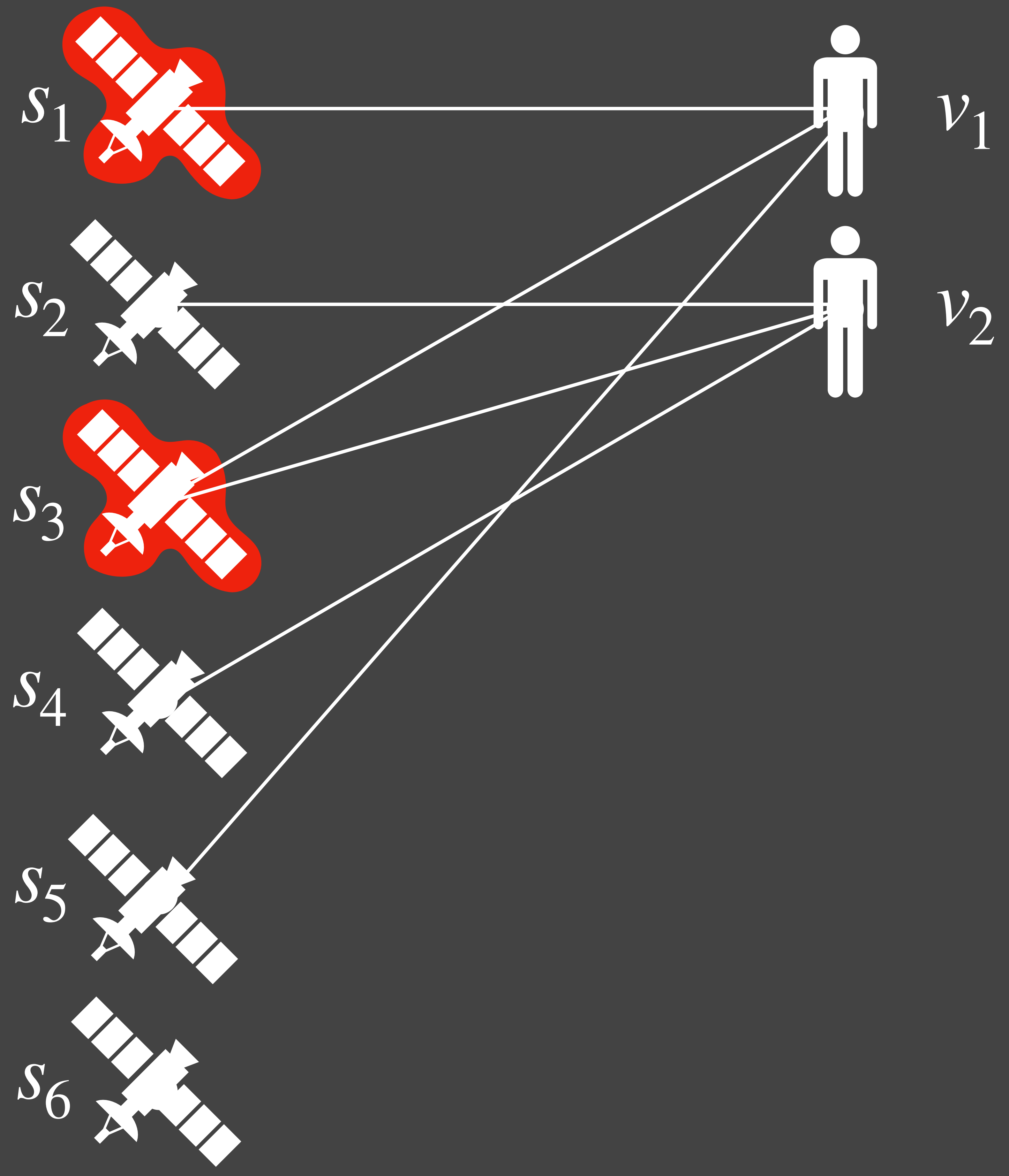
New model:
inserts AND deletes.

Moving to the Dynamic model



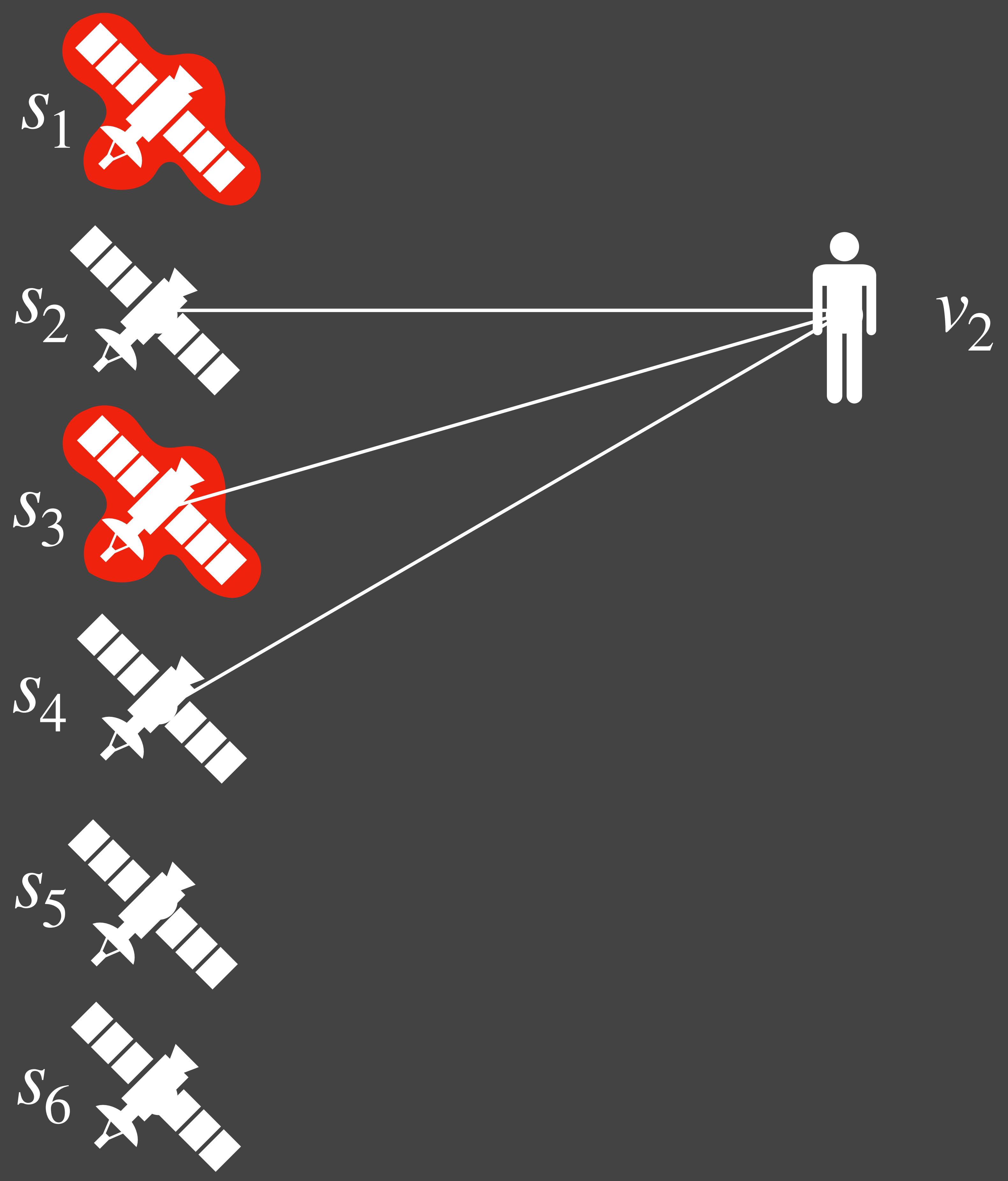
New model:
inserts AND deletes.

Moving to the Dynamic model



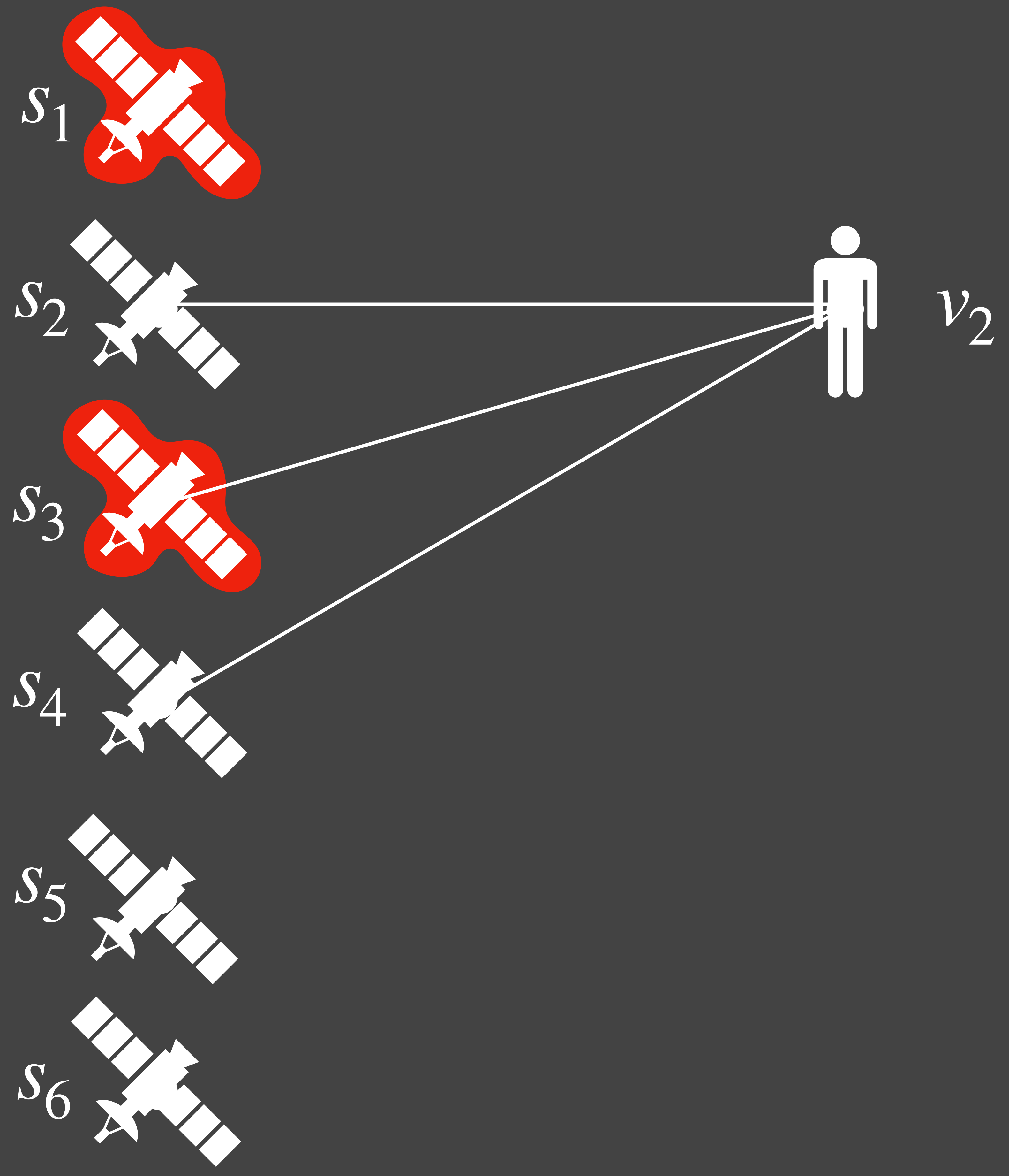
New model:
inserts AND deletes.

Moving to the Dynamic model



New model:
inserts AND deletes.

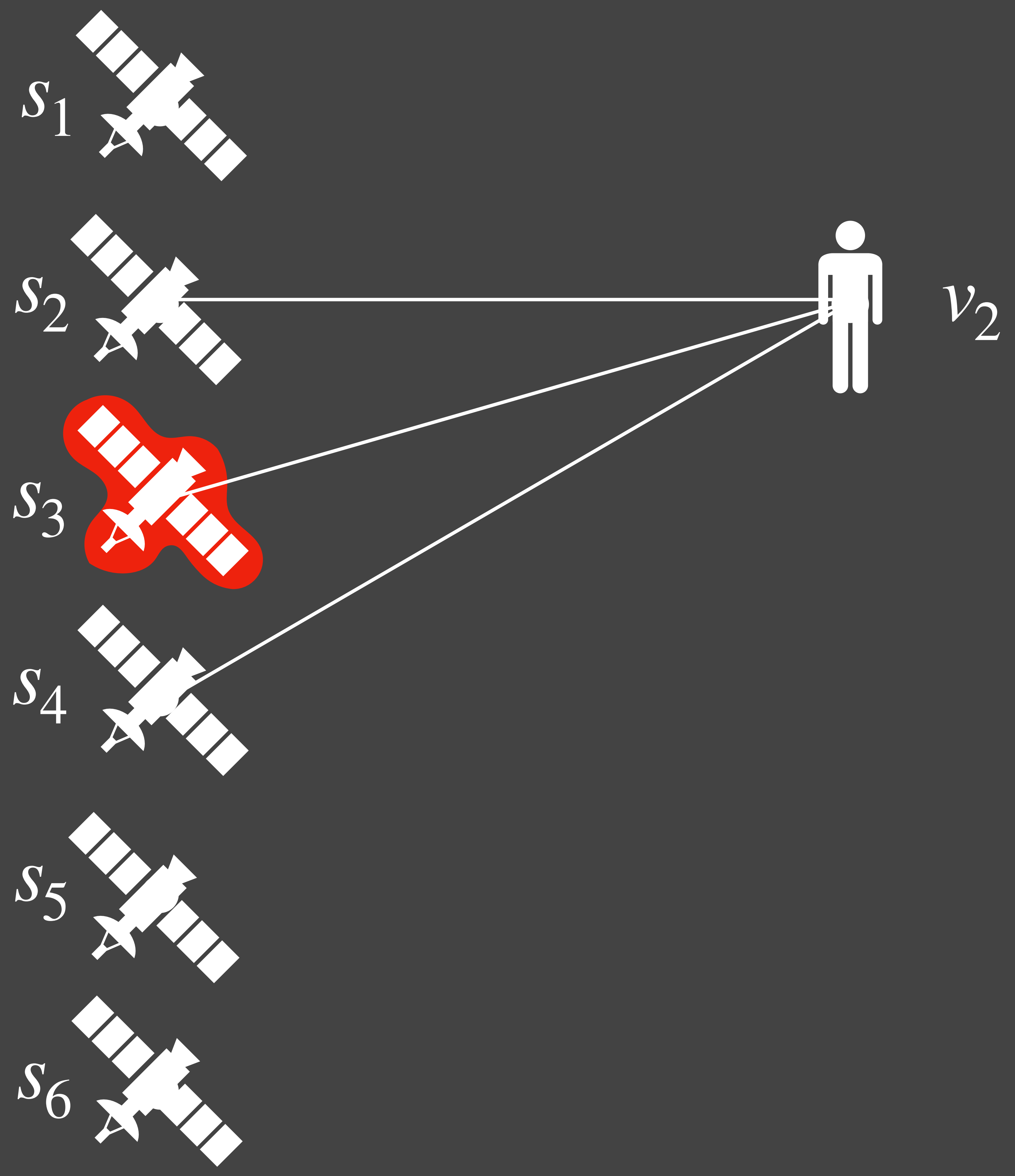
Moving to the Dynamic model



New model:
inserts AND deletes.

Algorithm now allowed limited # edits, a.k.a. recourse.

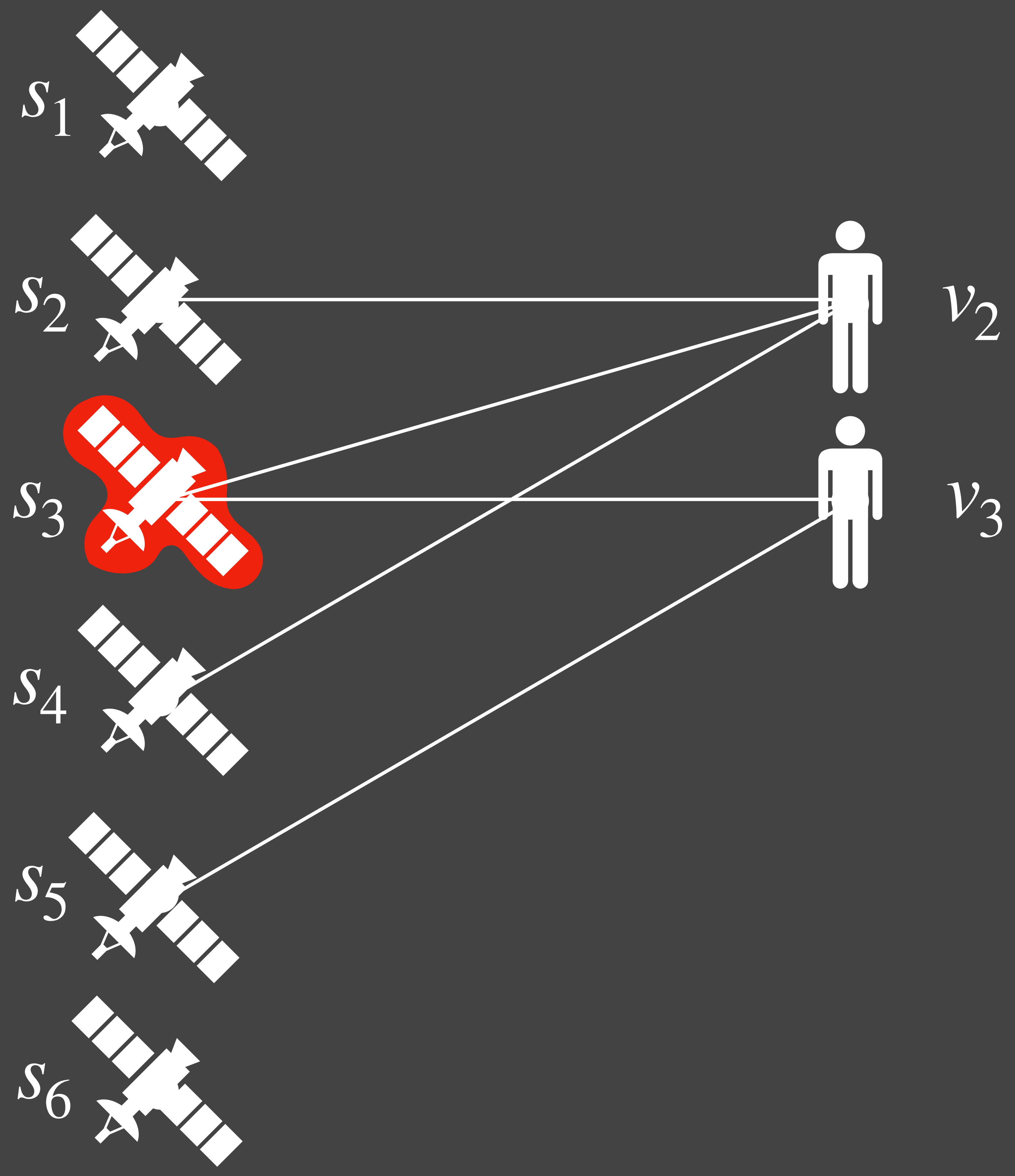
Moving to the Dynamic model



New model:
inserts AND deletes.

Algorithm now allowed limited # edits, a.k.a. recourse.

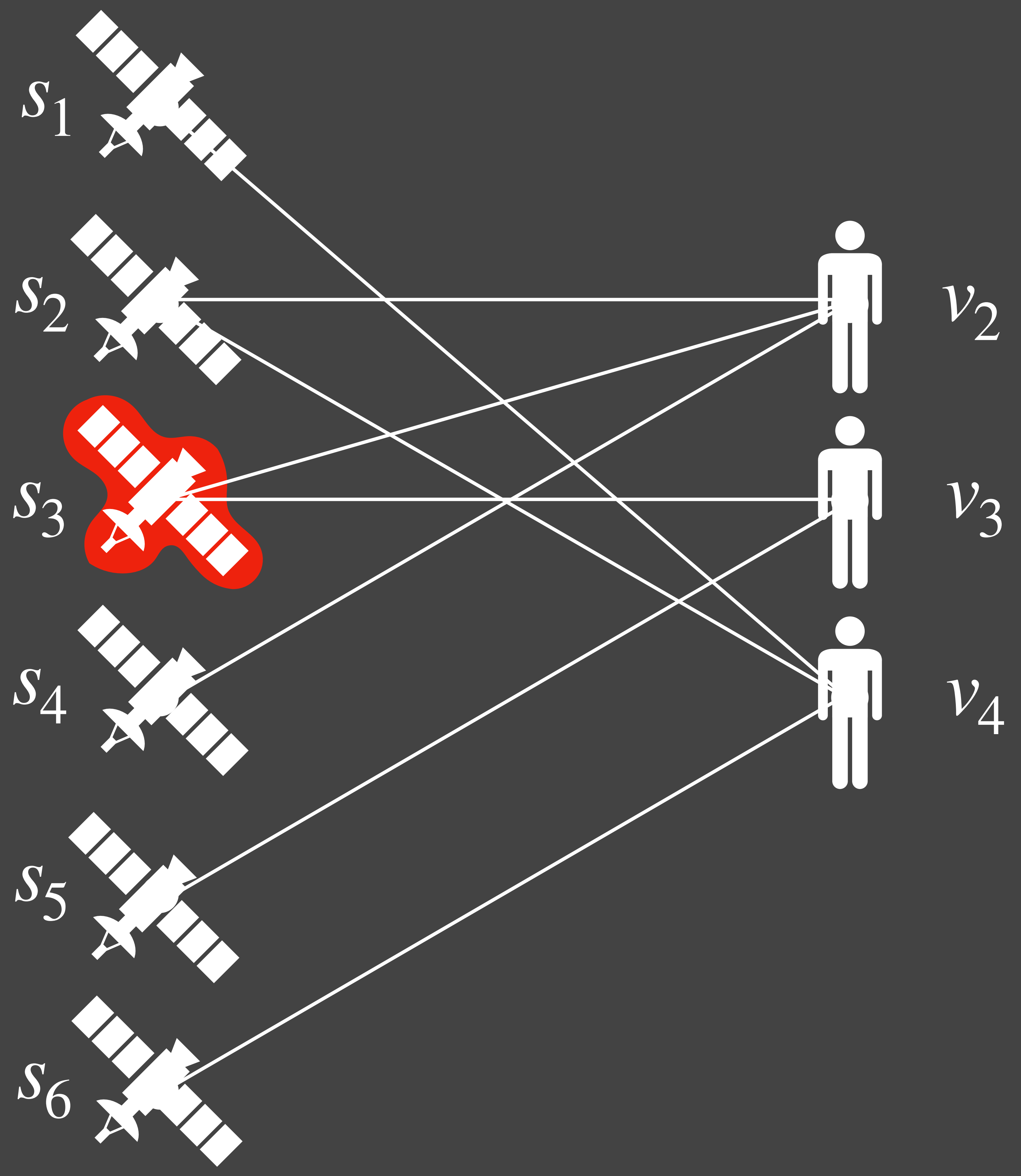
Moving to the Dynamic model



New model:
inserts AND deletes.

Algorithm now allowed limited # edits, a.k.a. recourse.

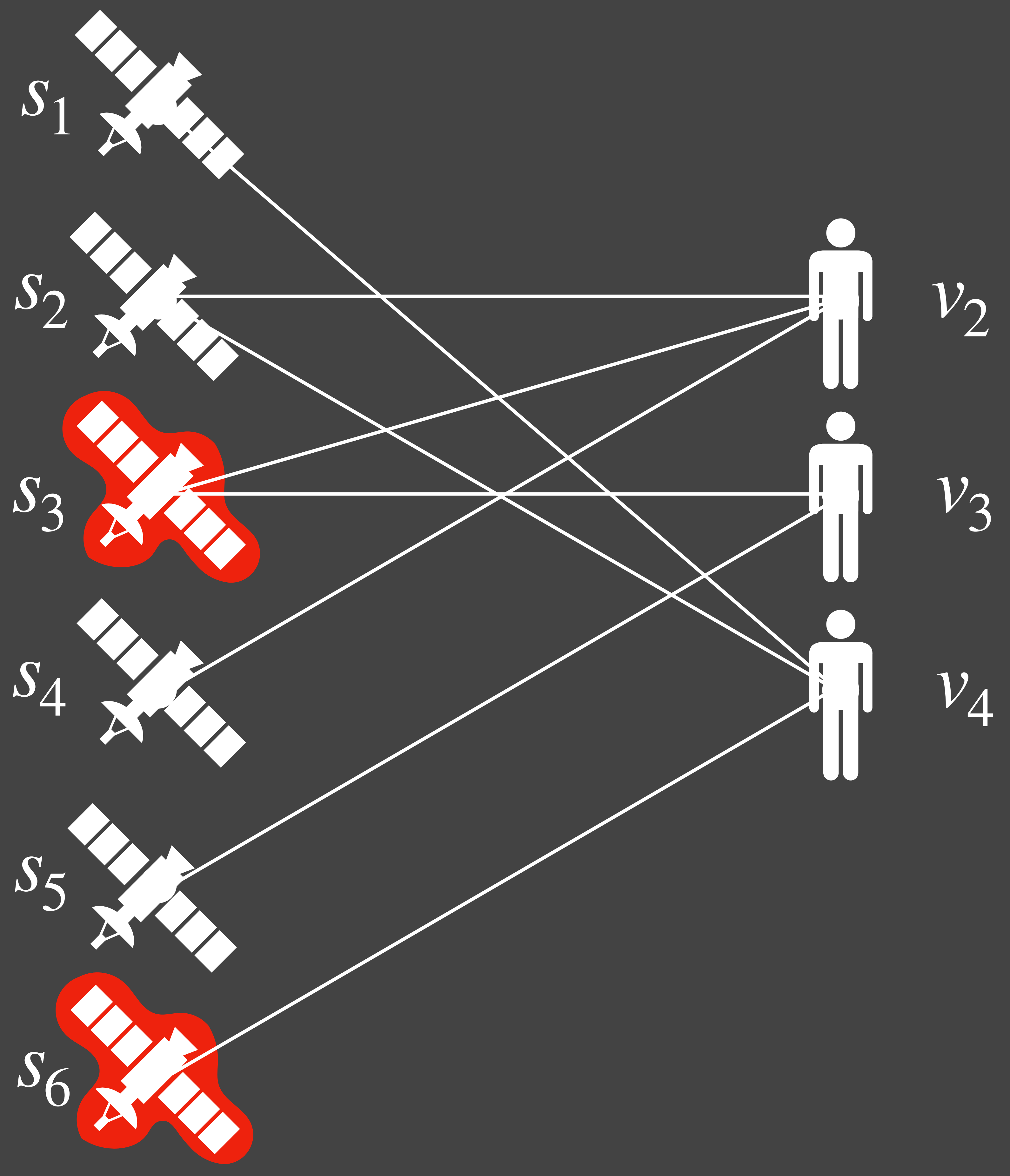
Moving to the Dynamic model



New model:
inserts AND deletes.

Algorithm now allowed limited # edits, a.k.a. recourse.

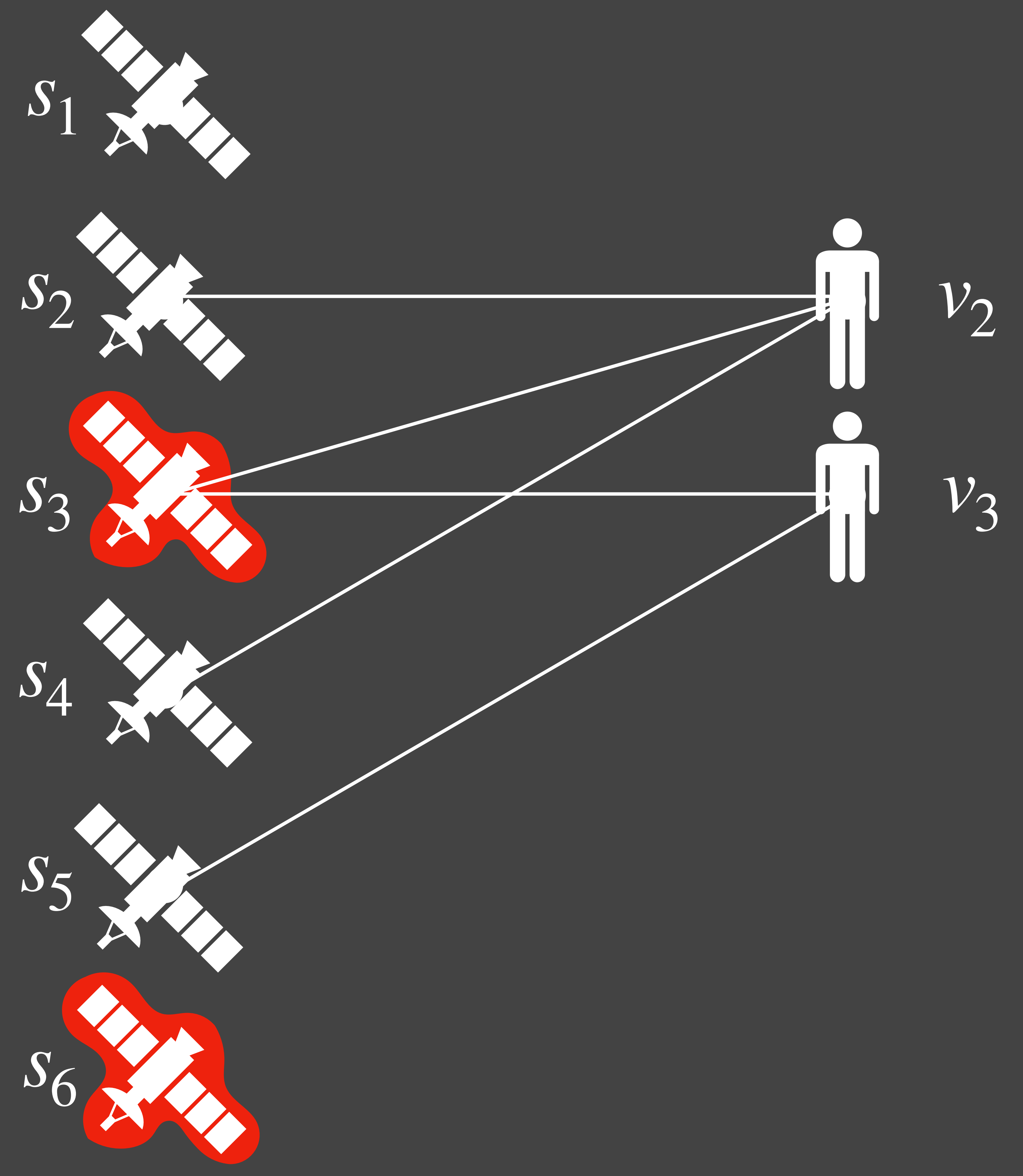
Moving to the Dynamic model



New model:
inserts AND deletes.

Algorithm now allowed limited # edits, a.k.a. recourse.

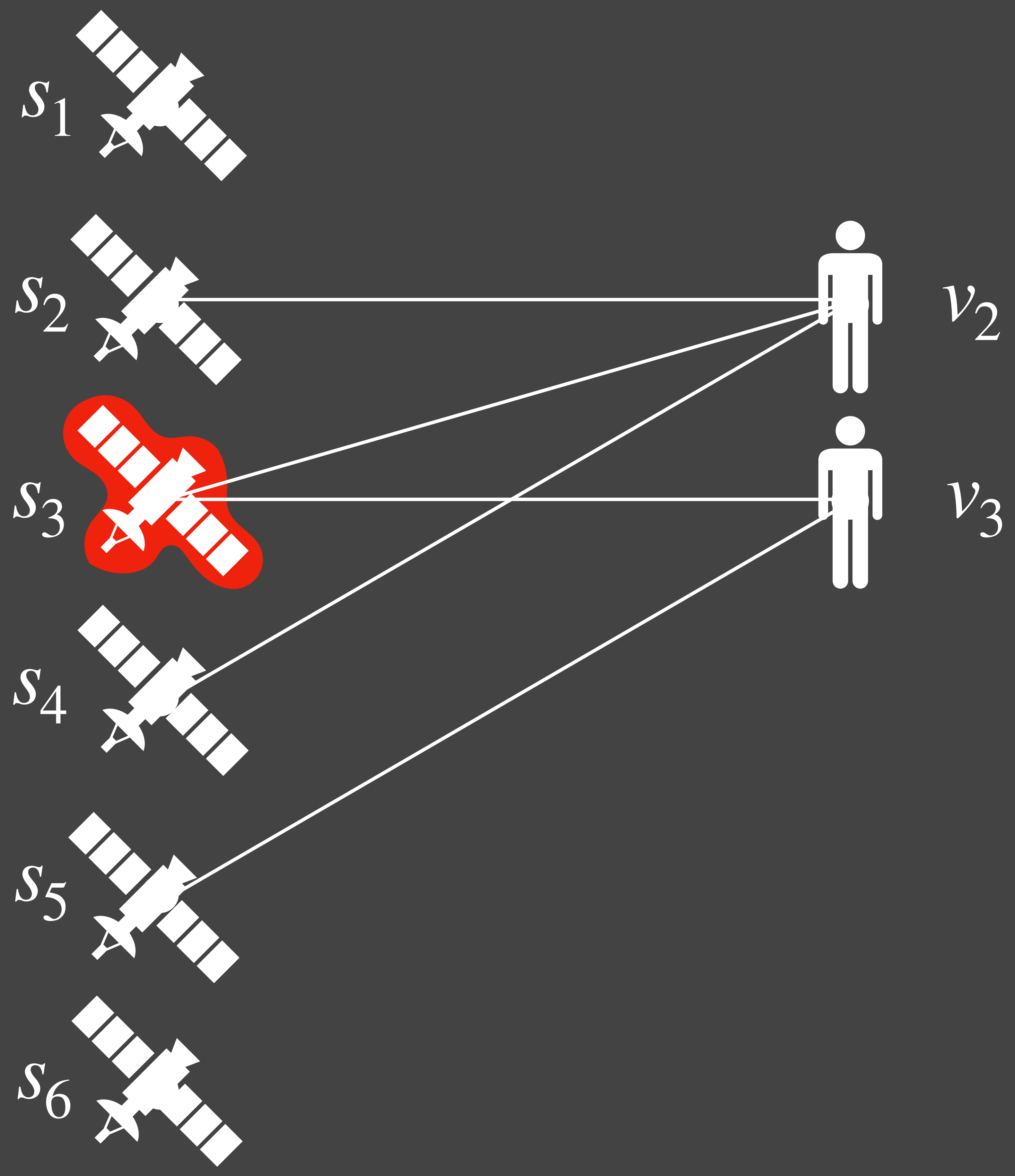
Moving to the Dynamic model



New model:
inserts AND deletes.

Algorithm now allowed limited # edits, a.k.a. recourse.

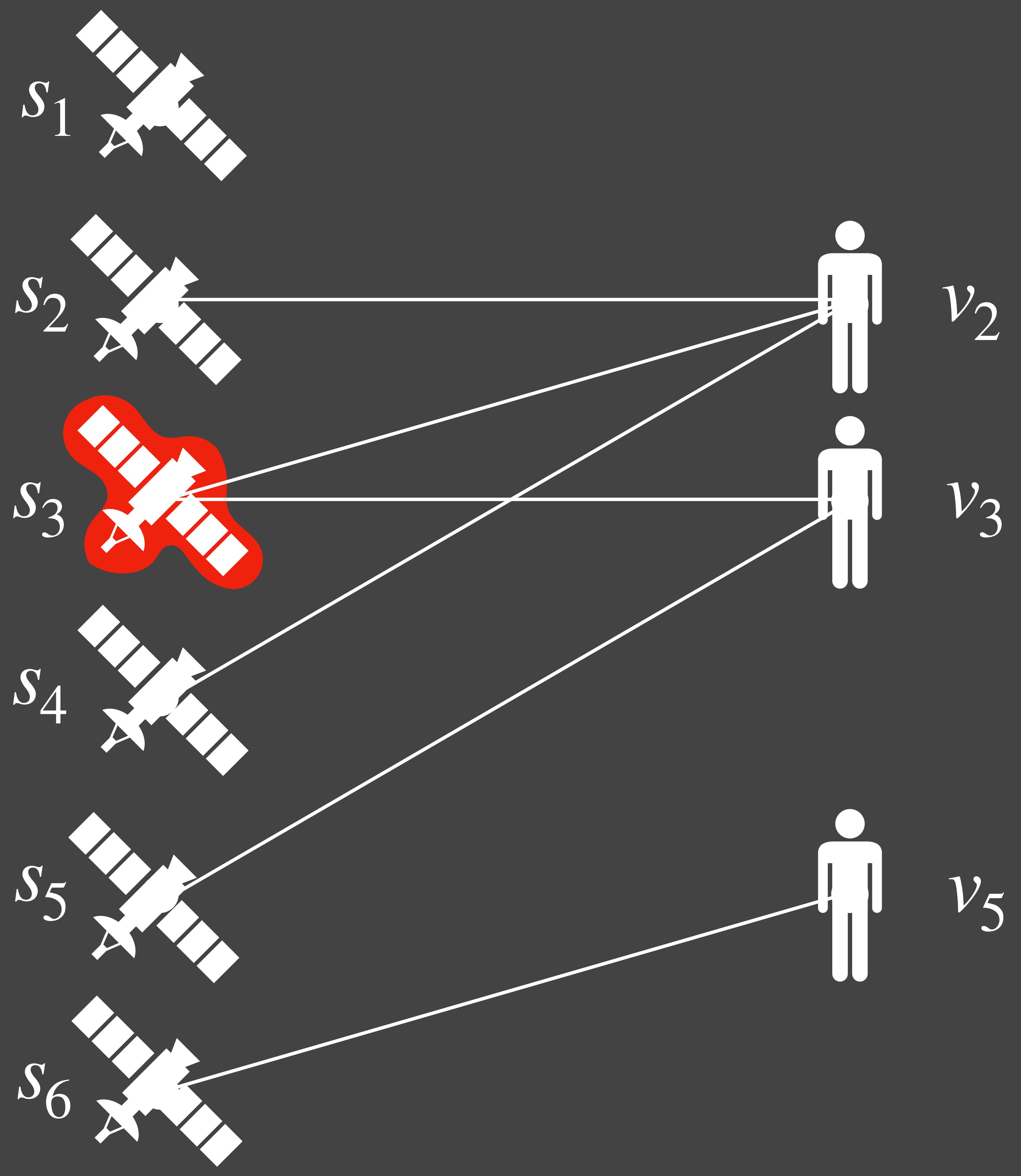
Moving to the Dynamic model



New model:
inserts AND deletes.

Algorithm now allowed limited # edits, a.k.a. recourse.

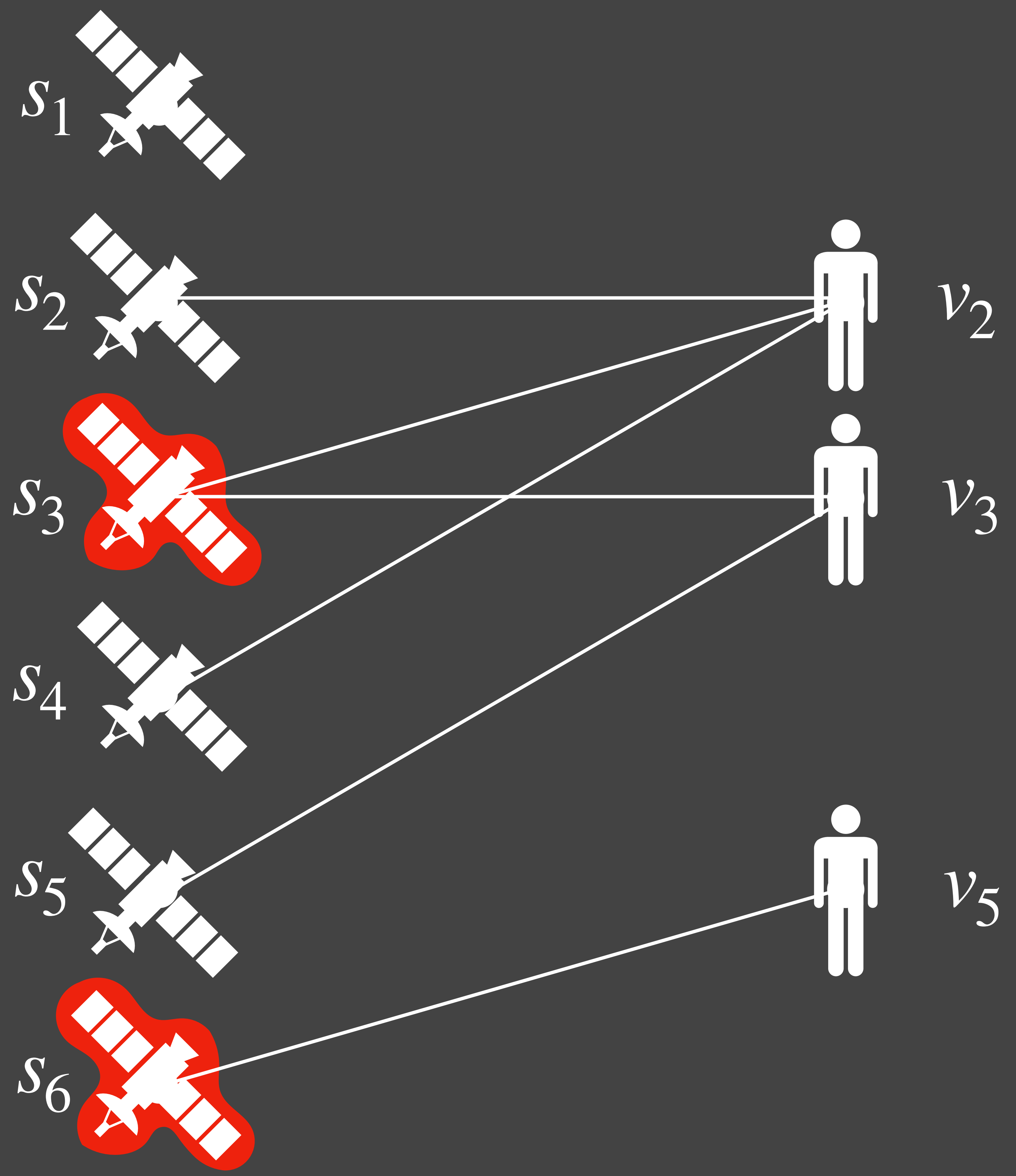
Moving to the Dynamic model



New model:
inserts AND deletes.

Algorithm now allowed limited # edits, a.k.a. recourse.

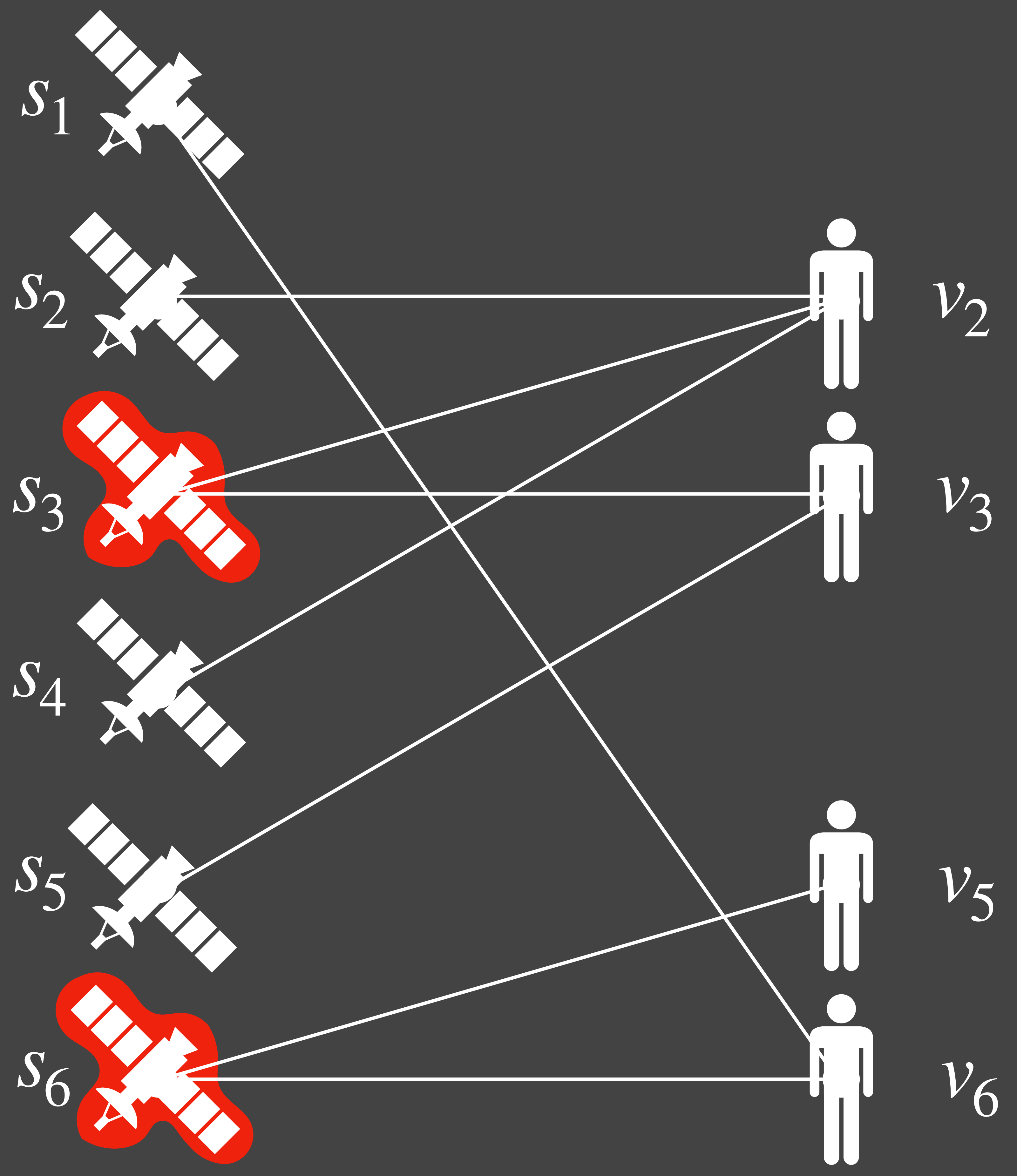
Moving to the Dynamic model



New model:
inserts AND deletes.

Algorithm now allowed limited # edits, a.k.a. recourse.

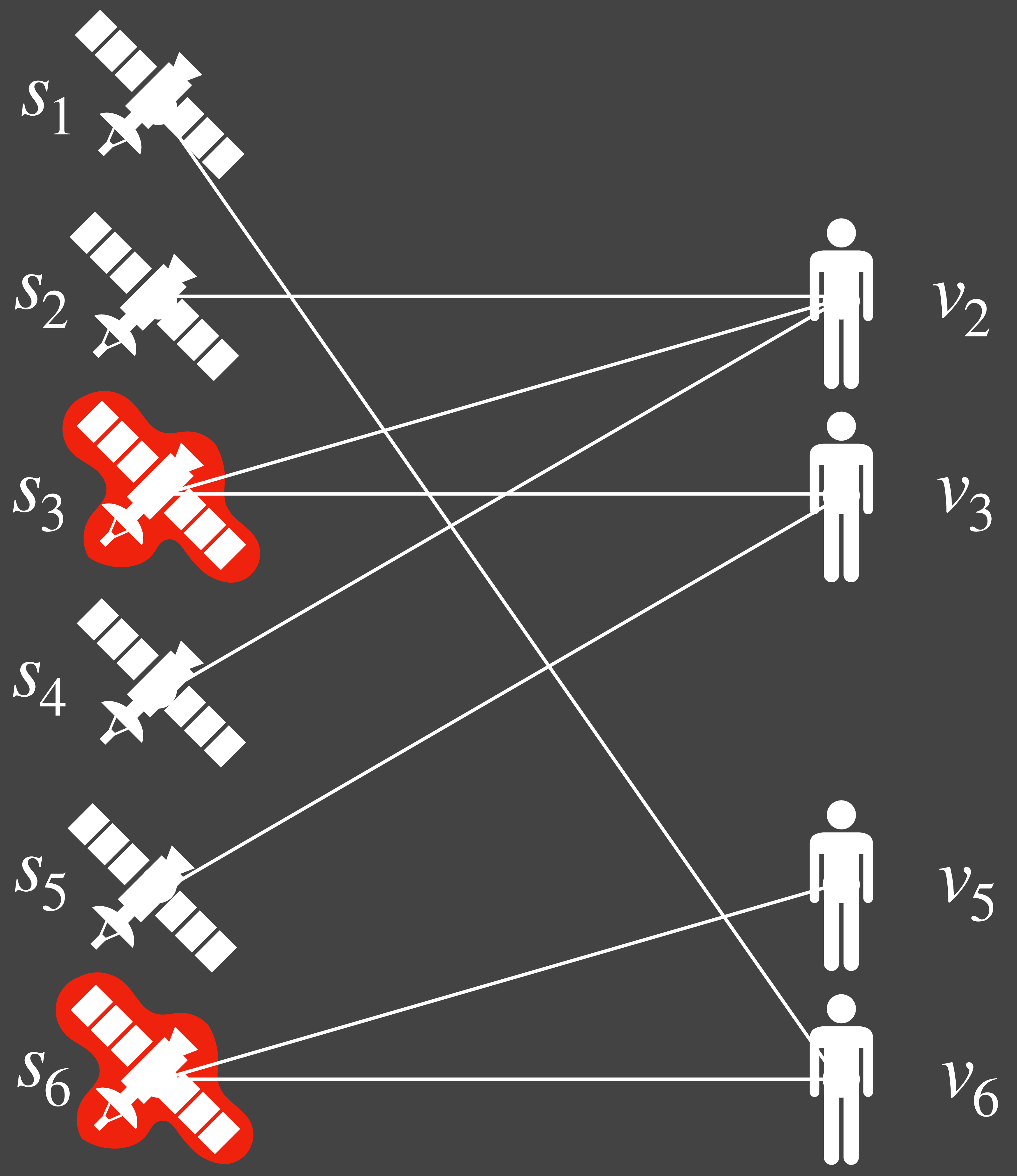
Moving to the Dynamic model



New model:
inserts AND deletes.

Algorithm now allowed limited # edits, a.k.a. recourse.

Moving to the Dynamic model

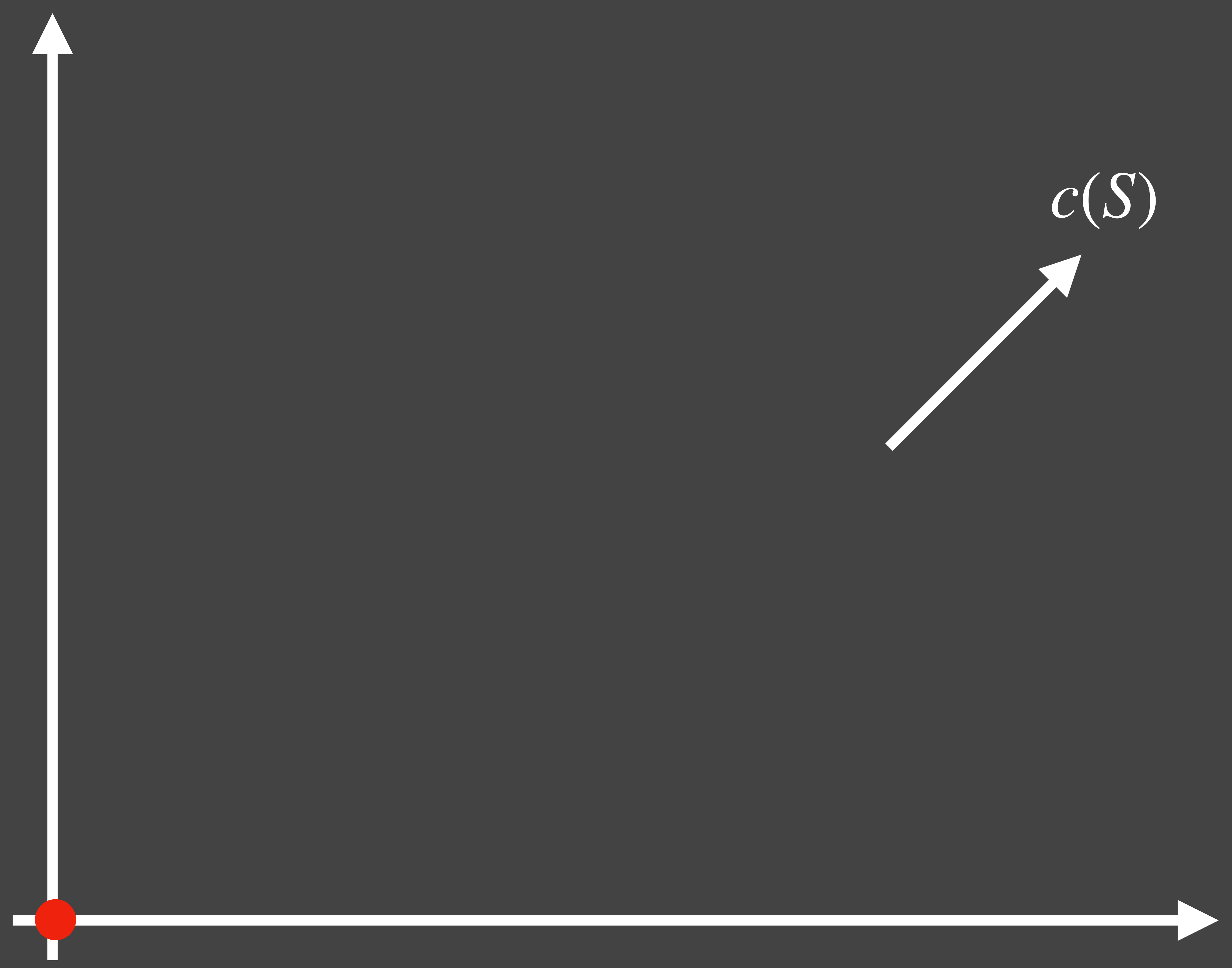


New model:
inserts AND deletes.

Algorithm now allowed limited # edits, a.k.a. recourse.

Q: Can we understand recourse/approximation tradeoffs?

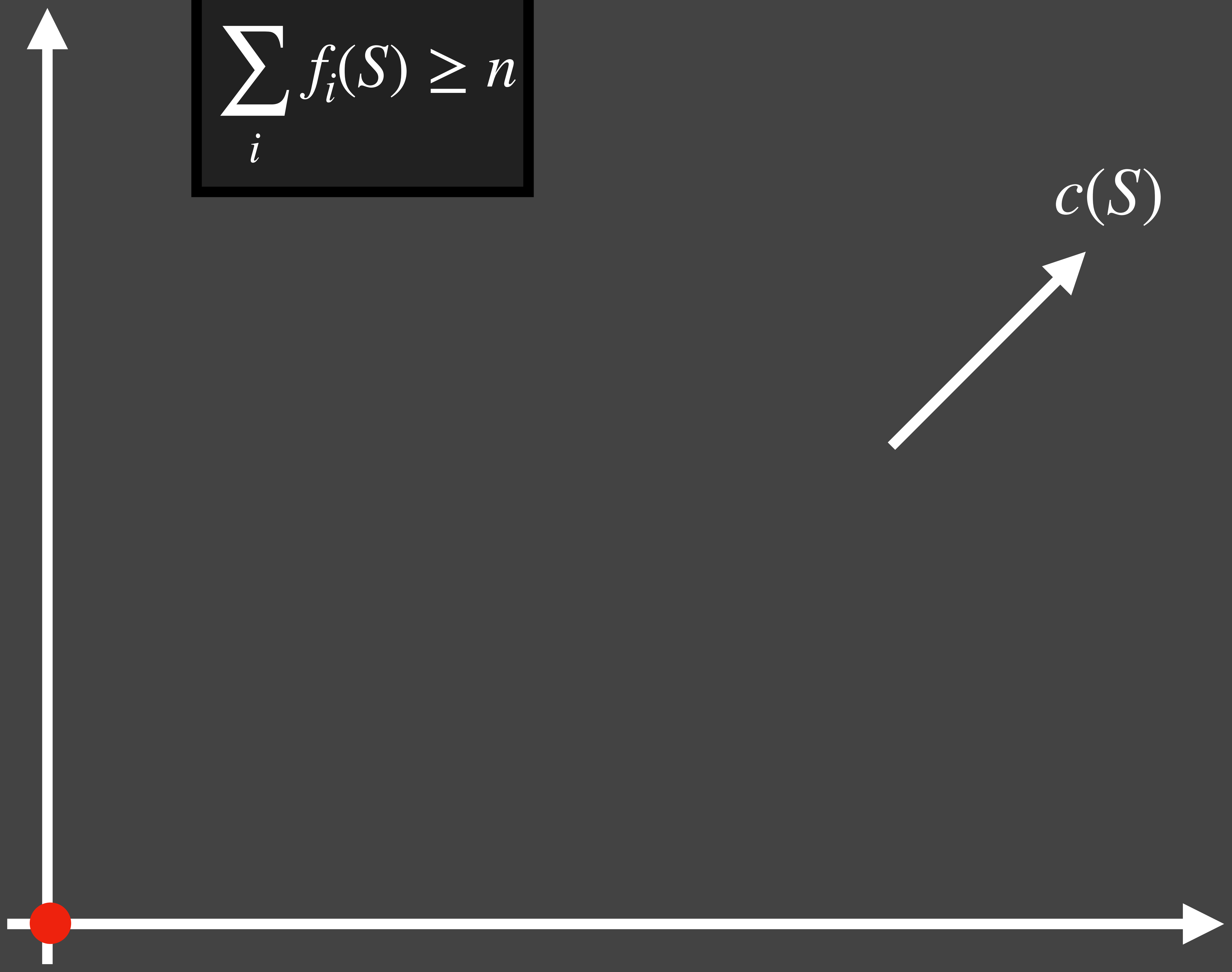
Dynamic Submodular Cover [Gupta L. FOCS 20]



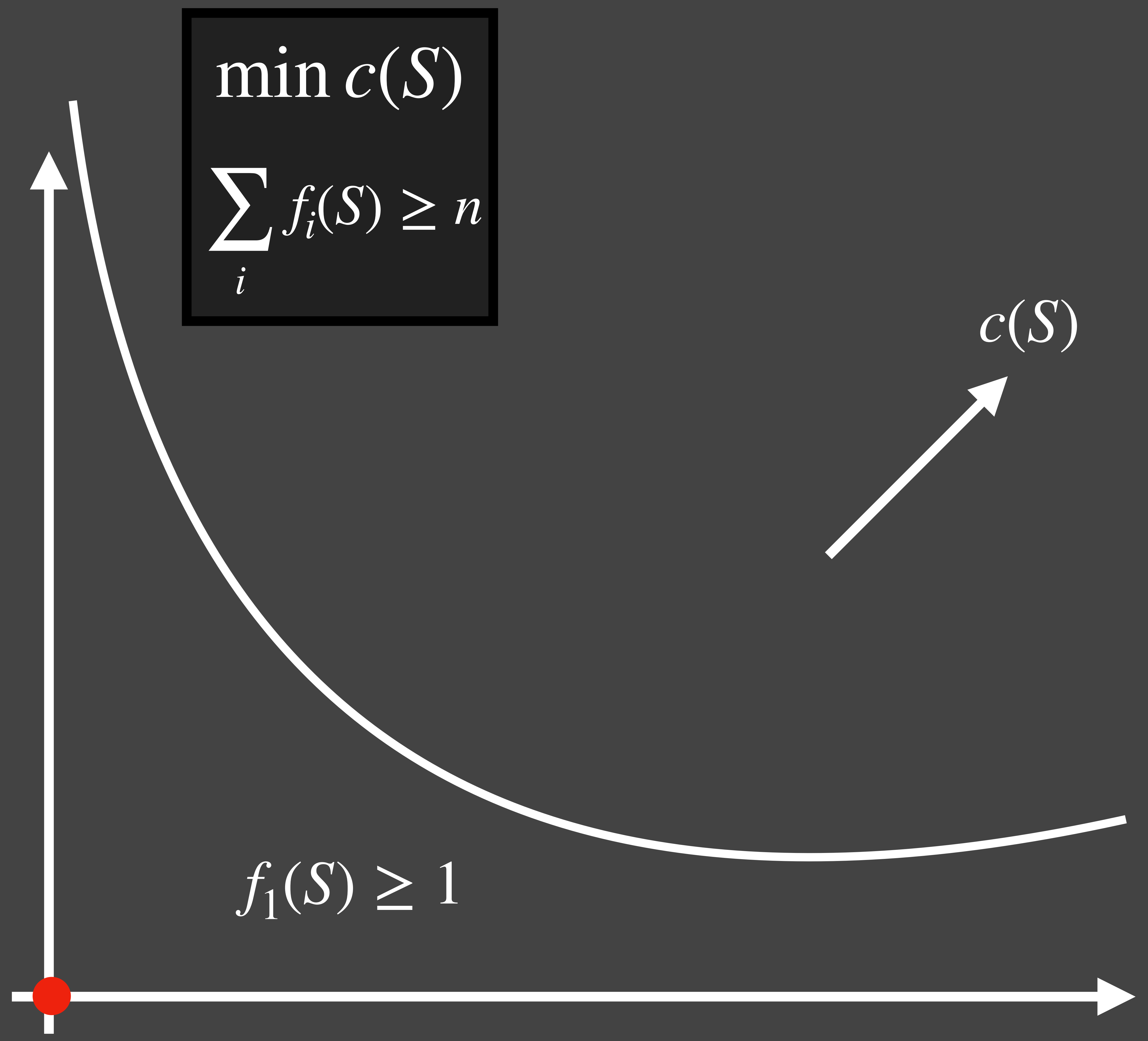
Dynamic Submodular Cover [Gupta L. FOCS 20]

$$\min c(S)$$

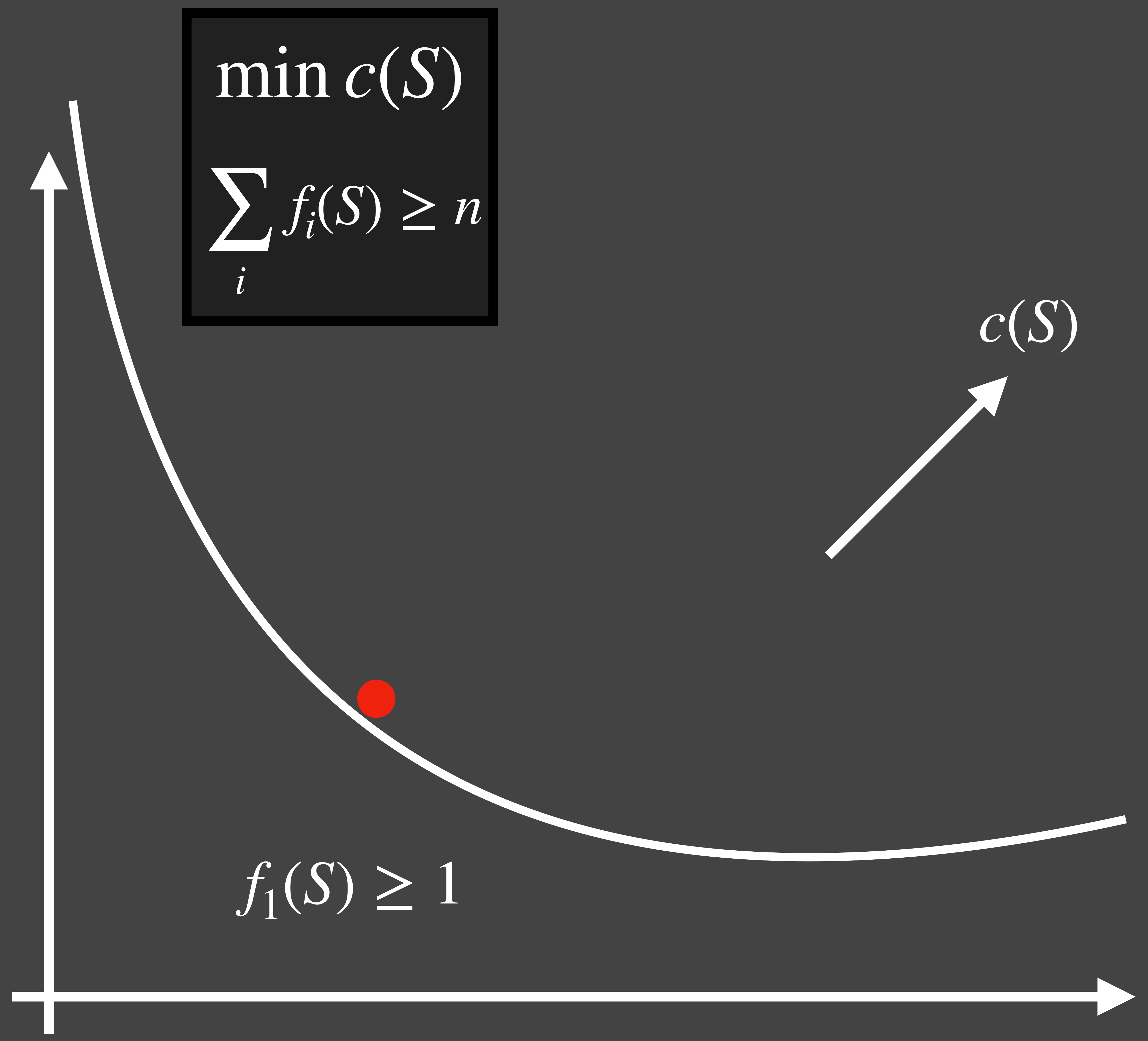
$$\sum_i f_i(S) \geq n$$



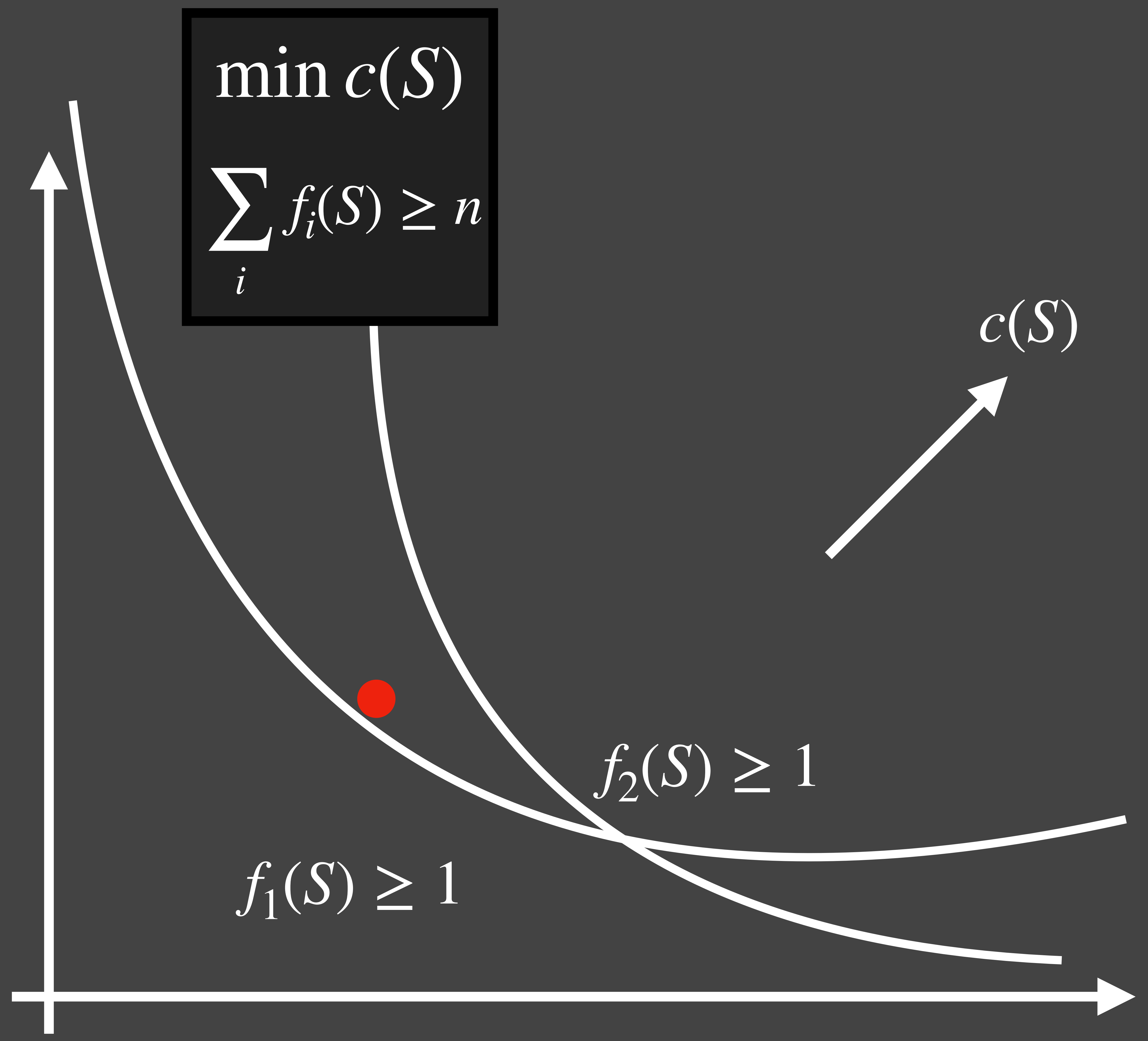
Dynamic Submodular Cover [Gupta L. FOCS 20]



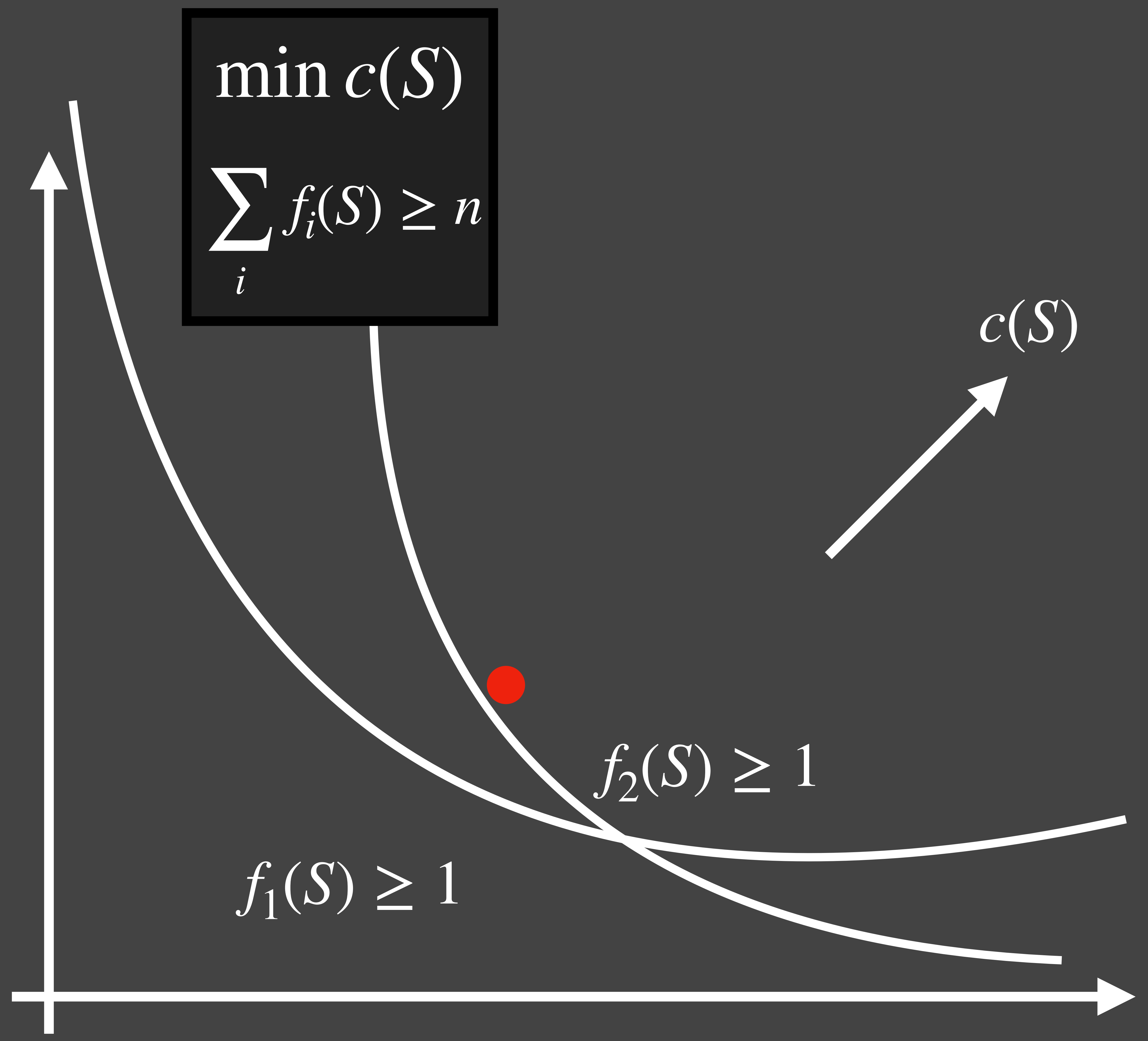
Dynamic Submodular Cover [Gupta L. FOCS 20]



Dynamic Submodular Cover [Gupta L. FOCS 20]

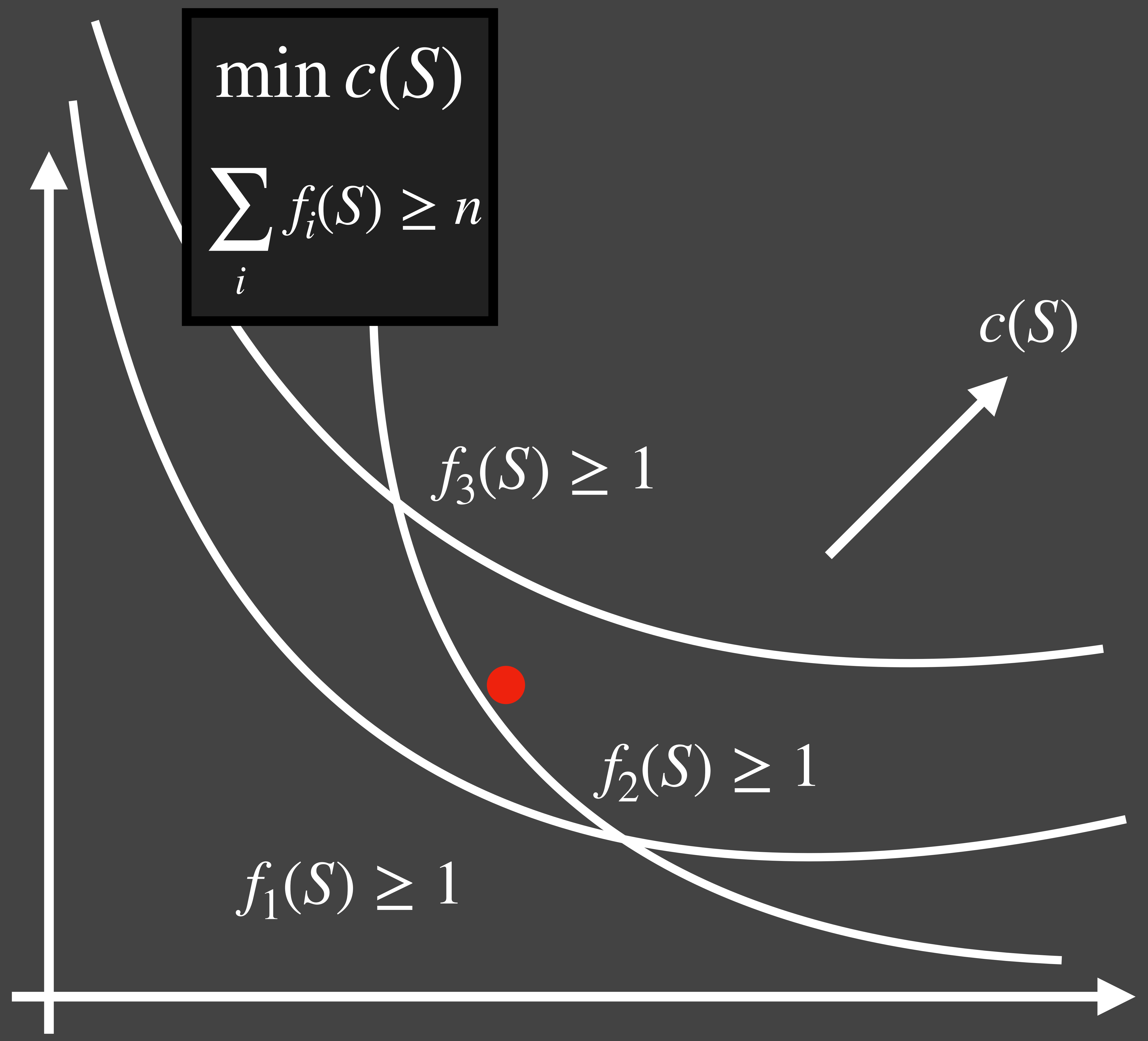


Dynamic Submodular Cover [Gupta L. FOCS 20]

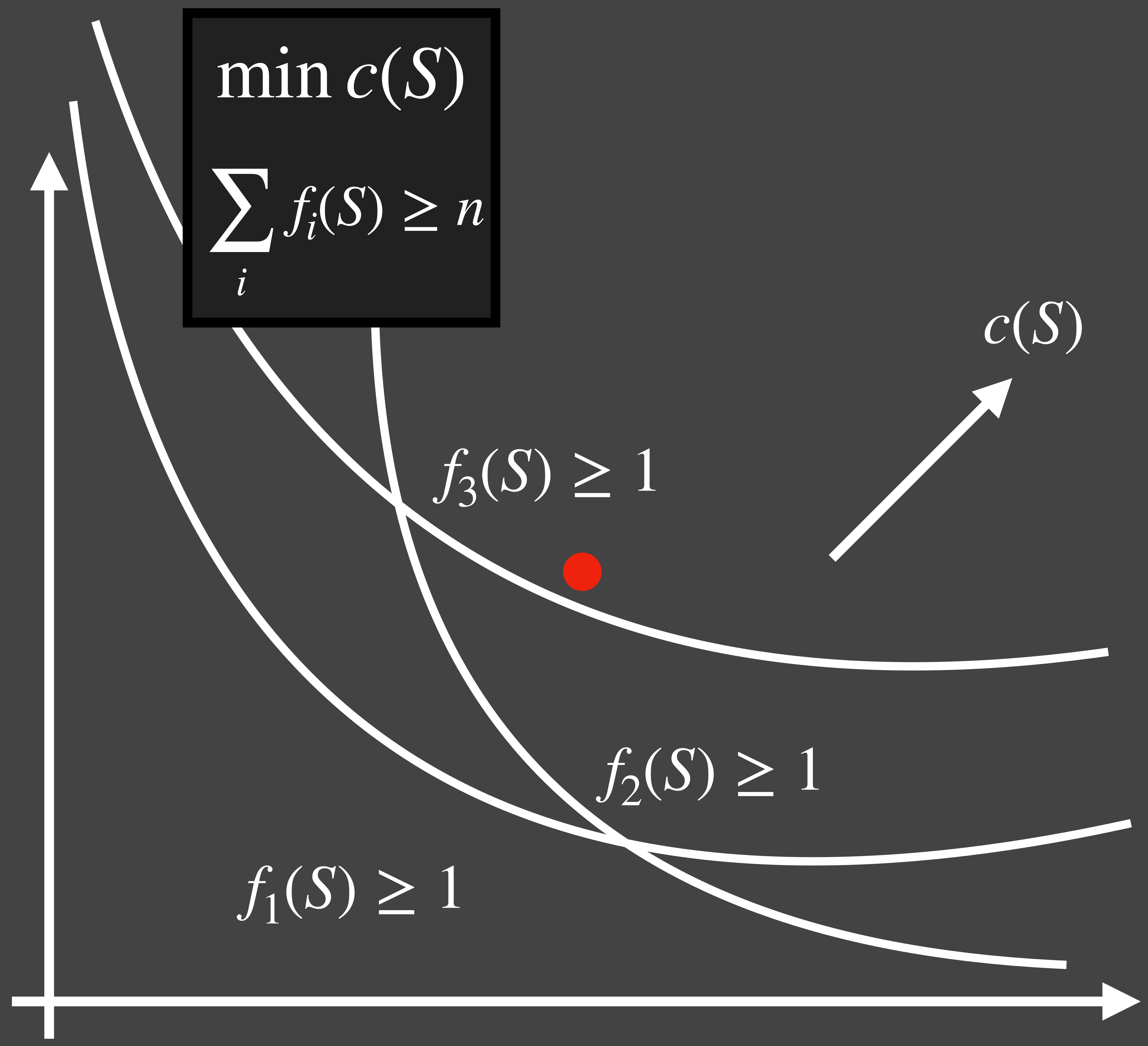


Dynamic Submodular Cover

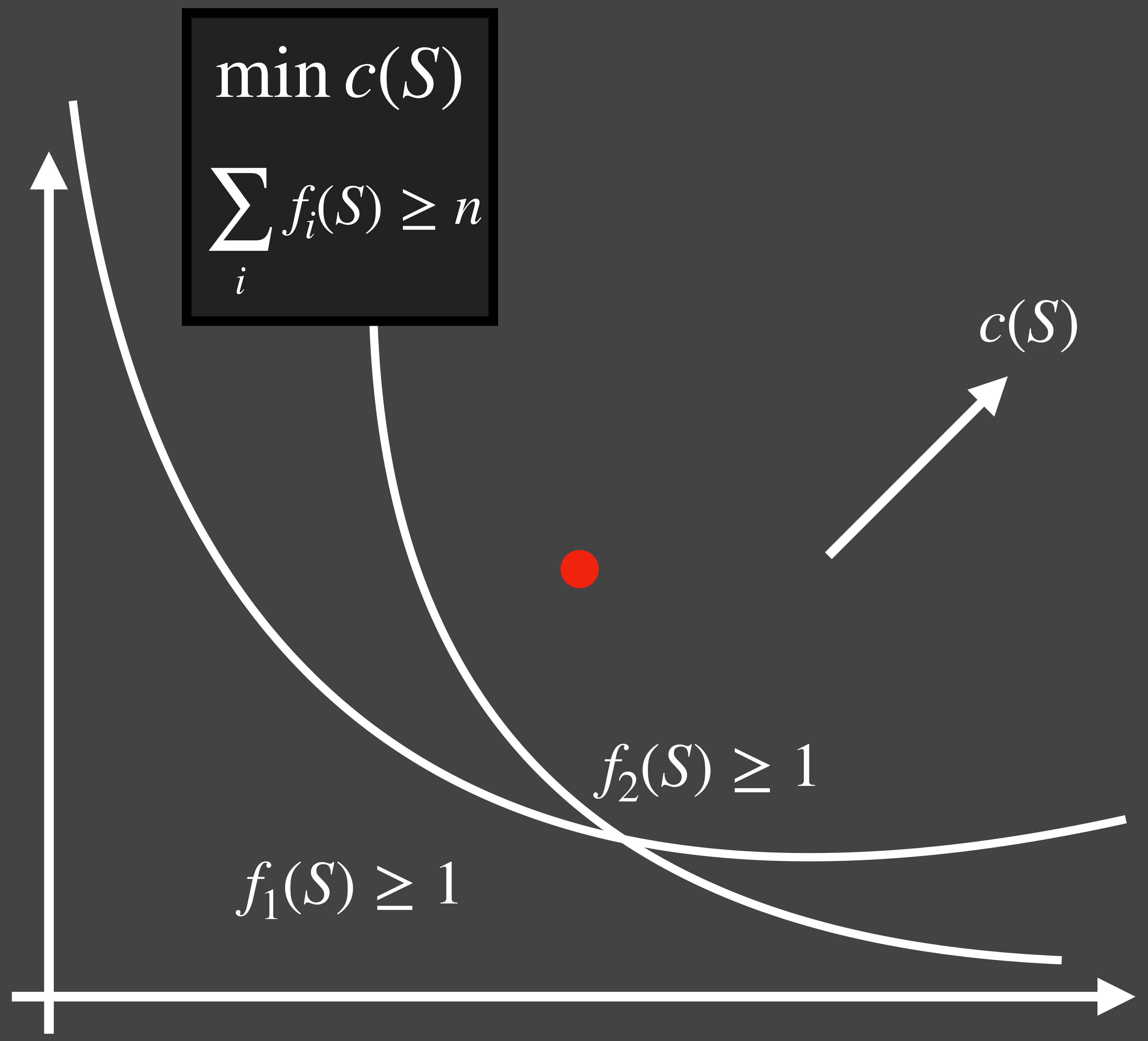
[Gupta L. FOCS 20]



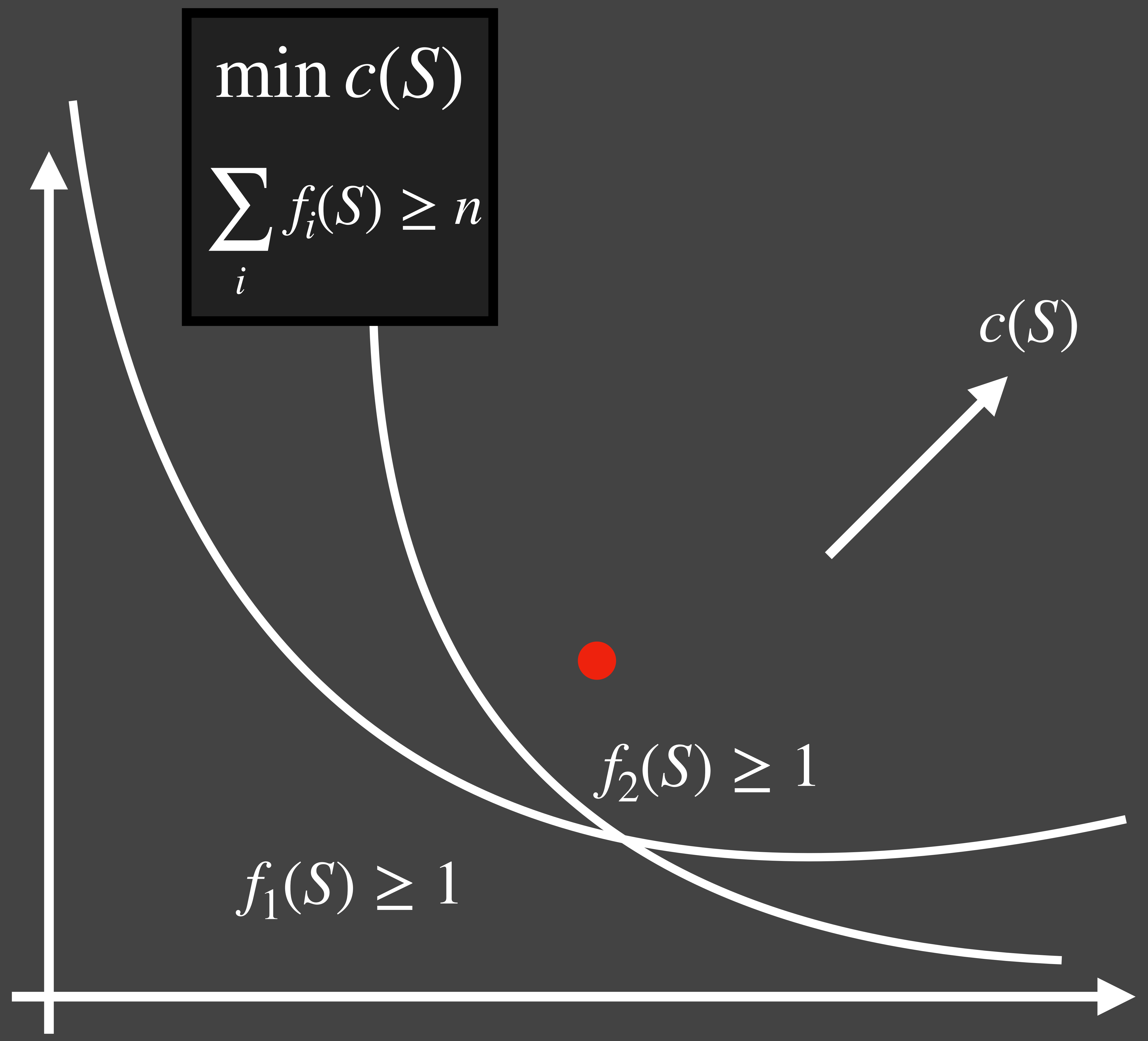
Dynamic Submodular Cover [Gupta L. FOCS 20]



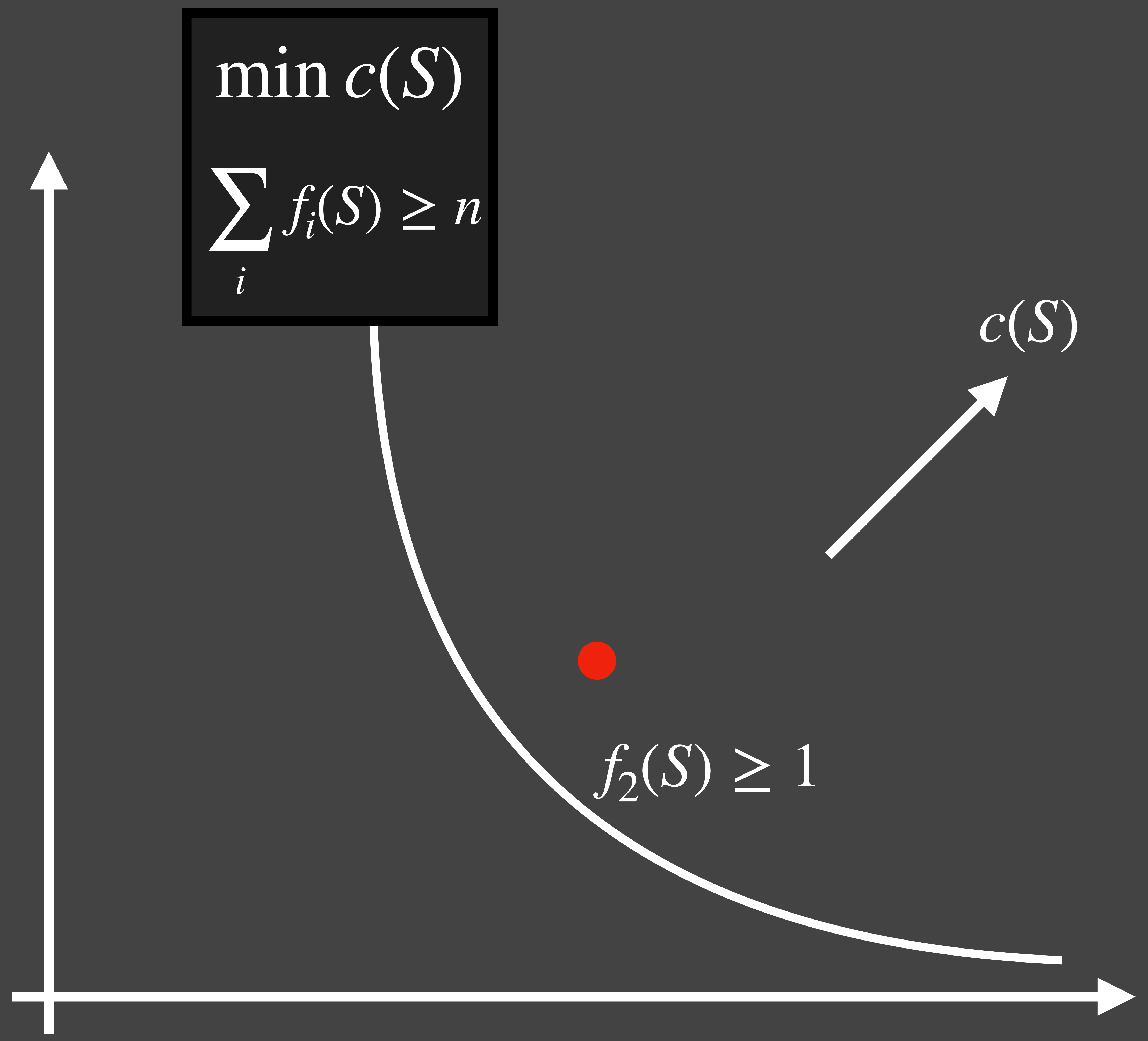
Dynamic Submodular Cover [Gupta L. FOCS 20]



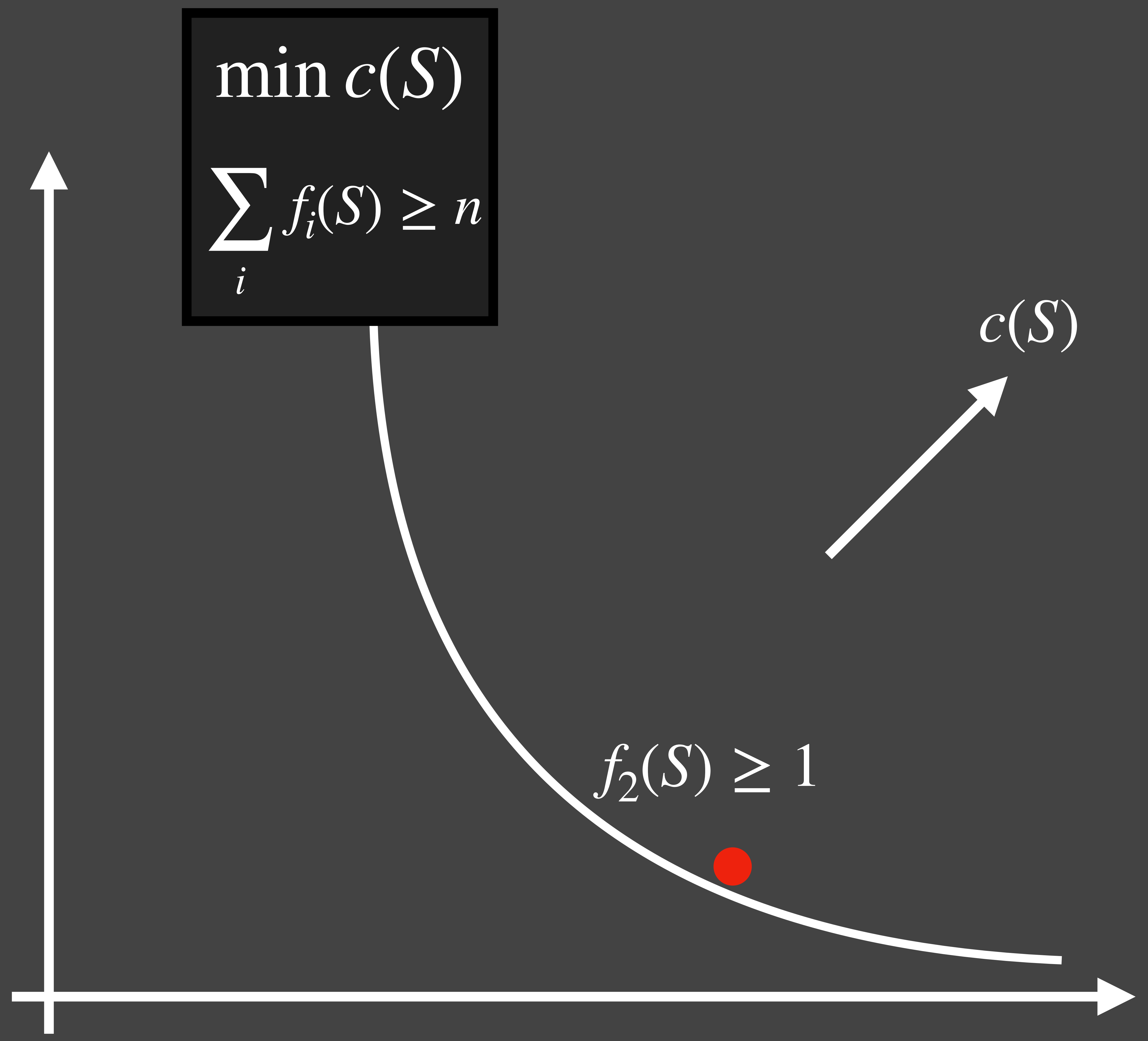
Dynamic Submodular Cover [Gupta L. FOCS 20]



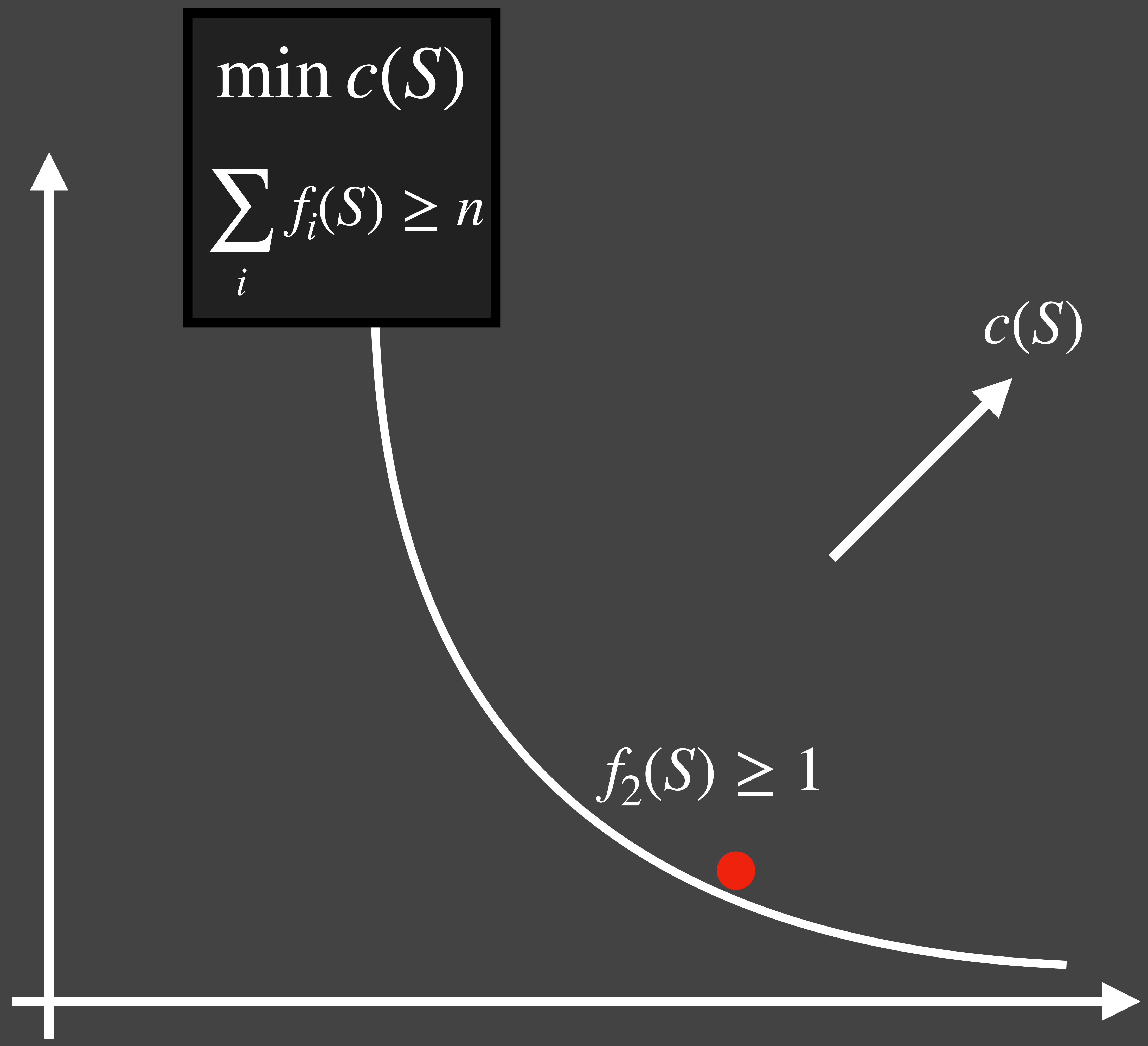
Dynamic Submodular Cover [Gupta L. FOCS 20]



Dynamic Submodular Cover [Gupta L. FOCS 20]



Dynamic Submodular Cover [Gupta L. FOCS 20]



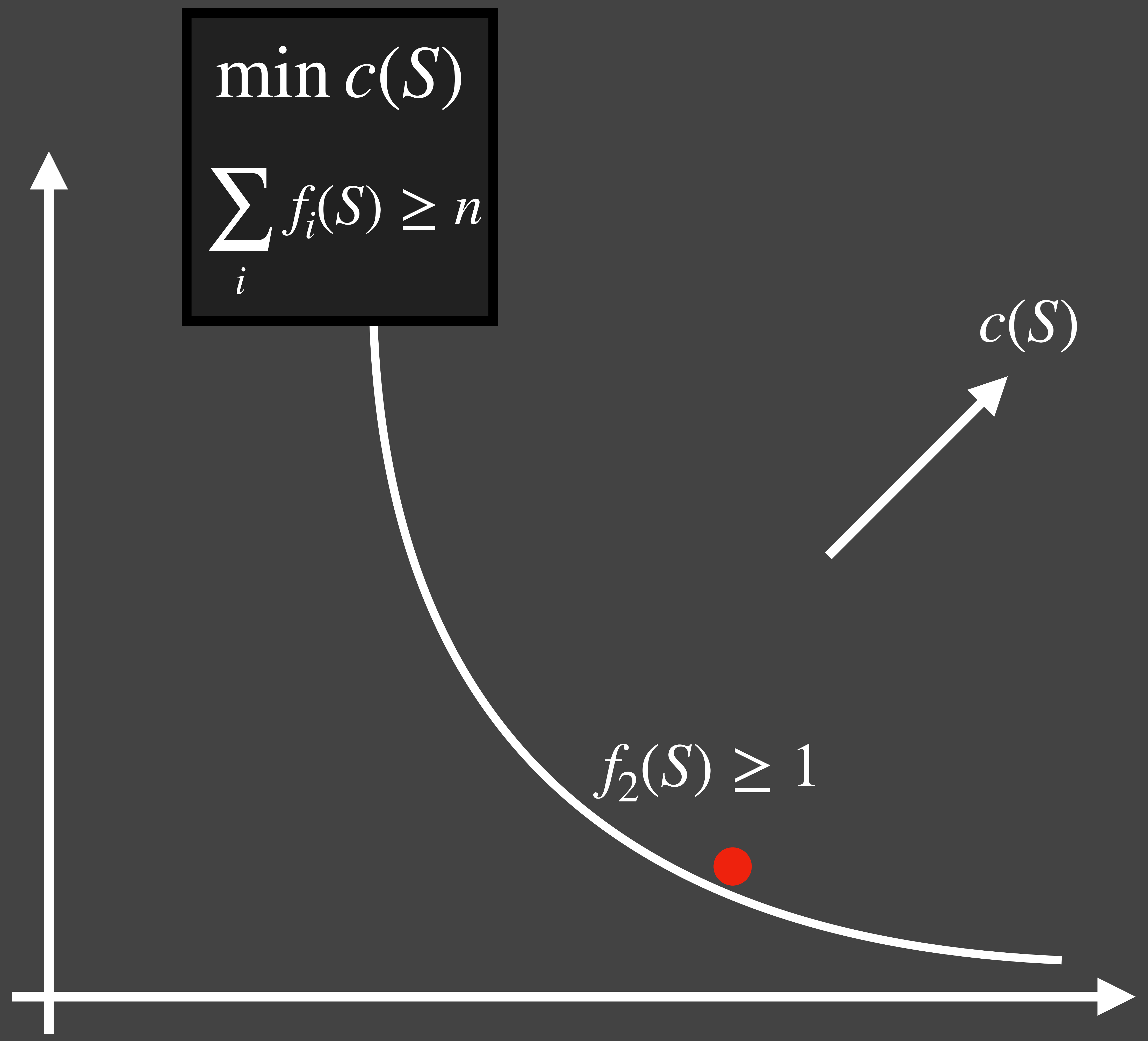
Theorem [Gupta L. FOCS 20]:

Polynomial time algo for Dynamic Submod Cover with:

(i) approximation $O(\log n)$.

(ii) recourse $\tilde{O}(1)$.

Dynamic Submodular Cover [Gupta L. FOCS 20]



Theorem [Gupta L. FOCS 20]:

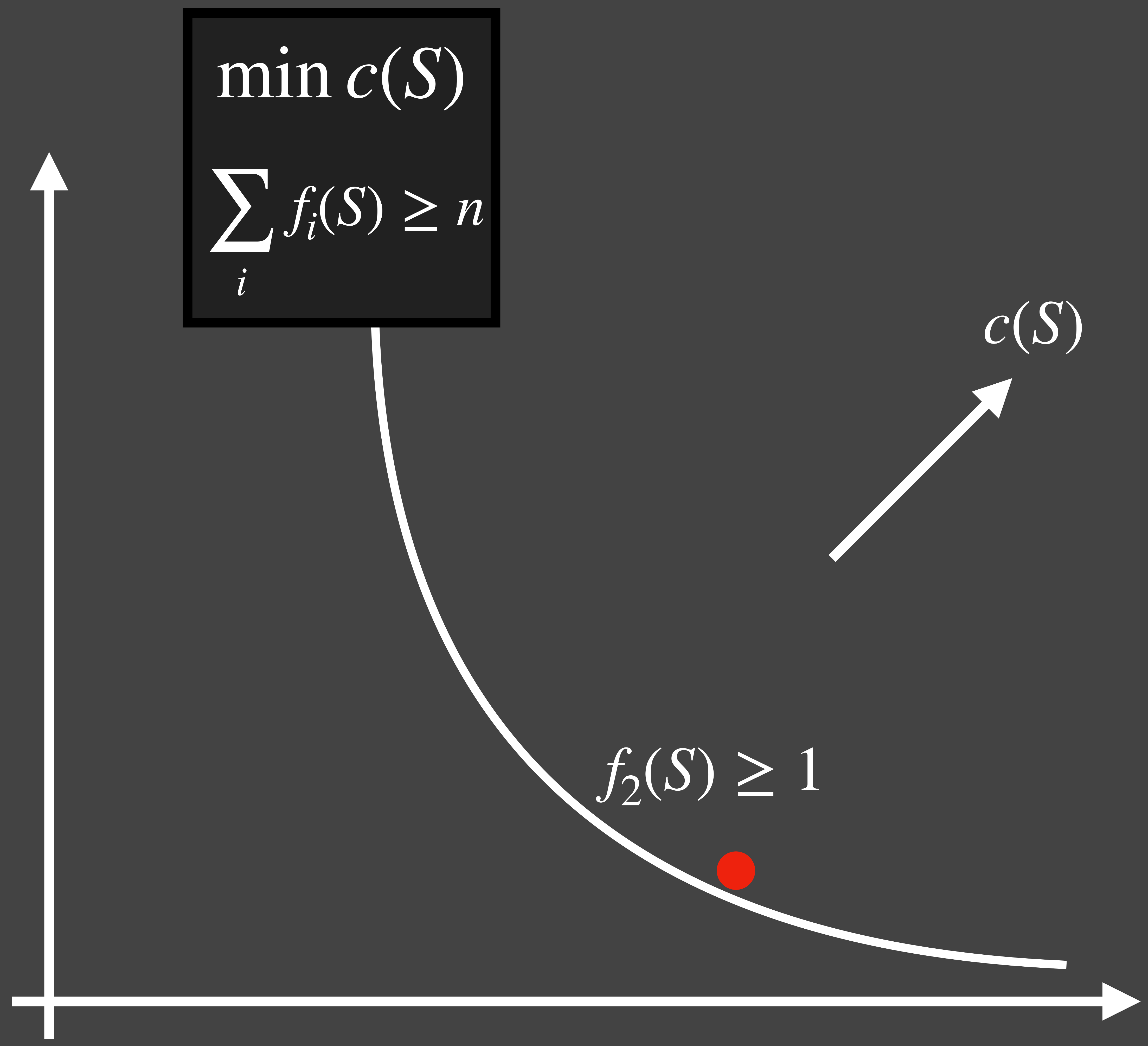
Polynomial time algo for Dynamic Submod Cover with:

(i) approximation $O(\log n)$.

(ii) recourse $\tilde{O}(1)$.

Optimal!

Dynamic Submodular Cover [Gupta L. FOCS 20]



Theorem [Gupta L. FOCS 20]:

Polynomial time algo for Dynamic Submod Cover with:

(i) approximation $O(\log n)$.

(ii) recourse $\tilde{O}(1)$.

Optimal!

Technical Ingredient:
*Template for converting **greedy** algos to **local search** algos, + **Tsallis Entropy** potential for analysis!*

Comparison

Online

- Inserts Only
- Decisions are **irrevocable**

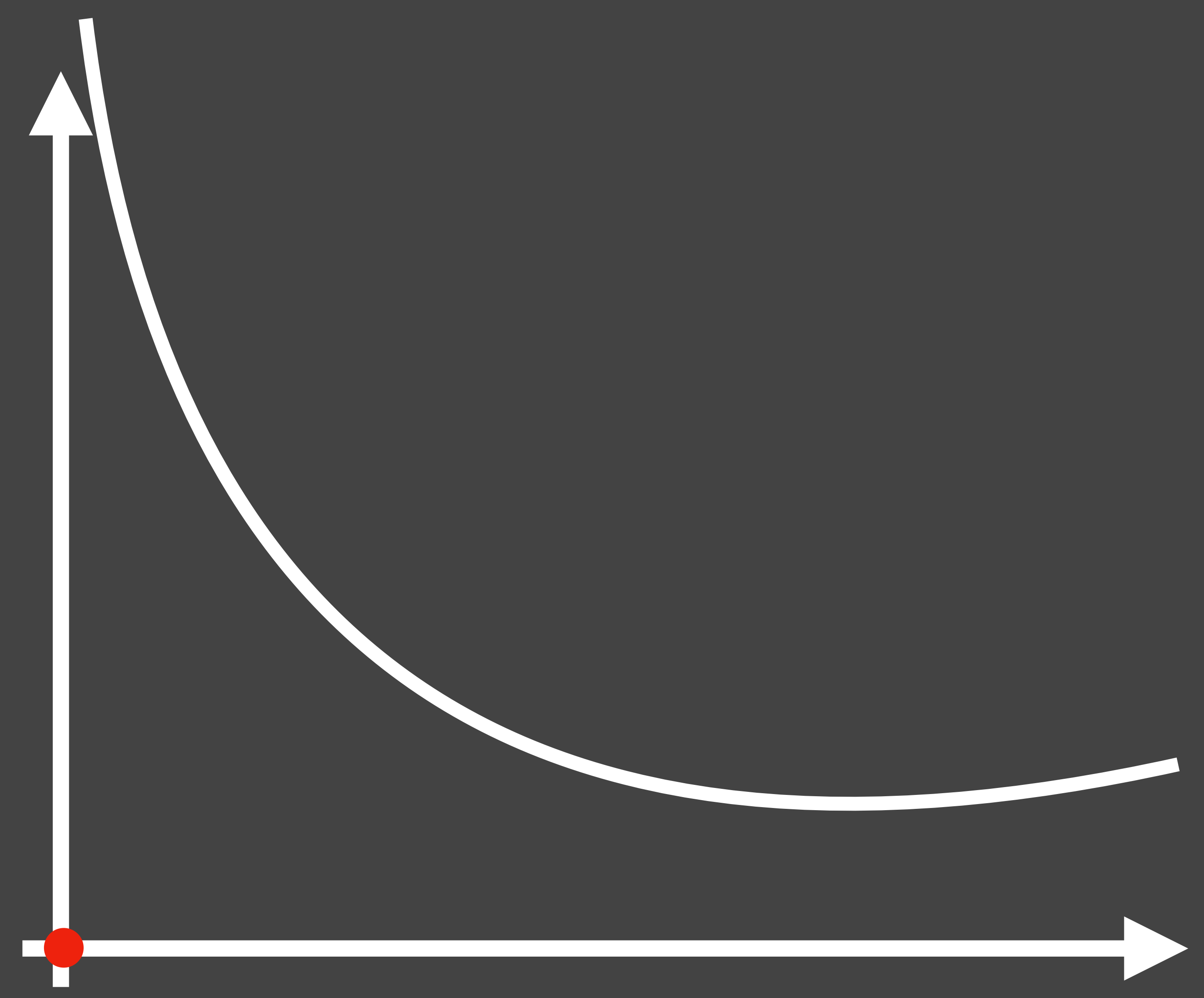
Dynamic

- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**

Comparison

Online

- Inserts Only
- Decisions are **irrevocable**



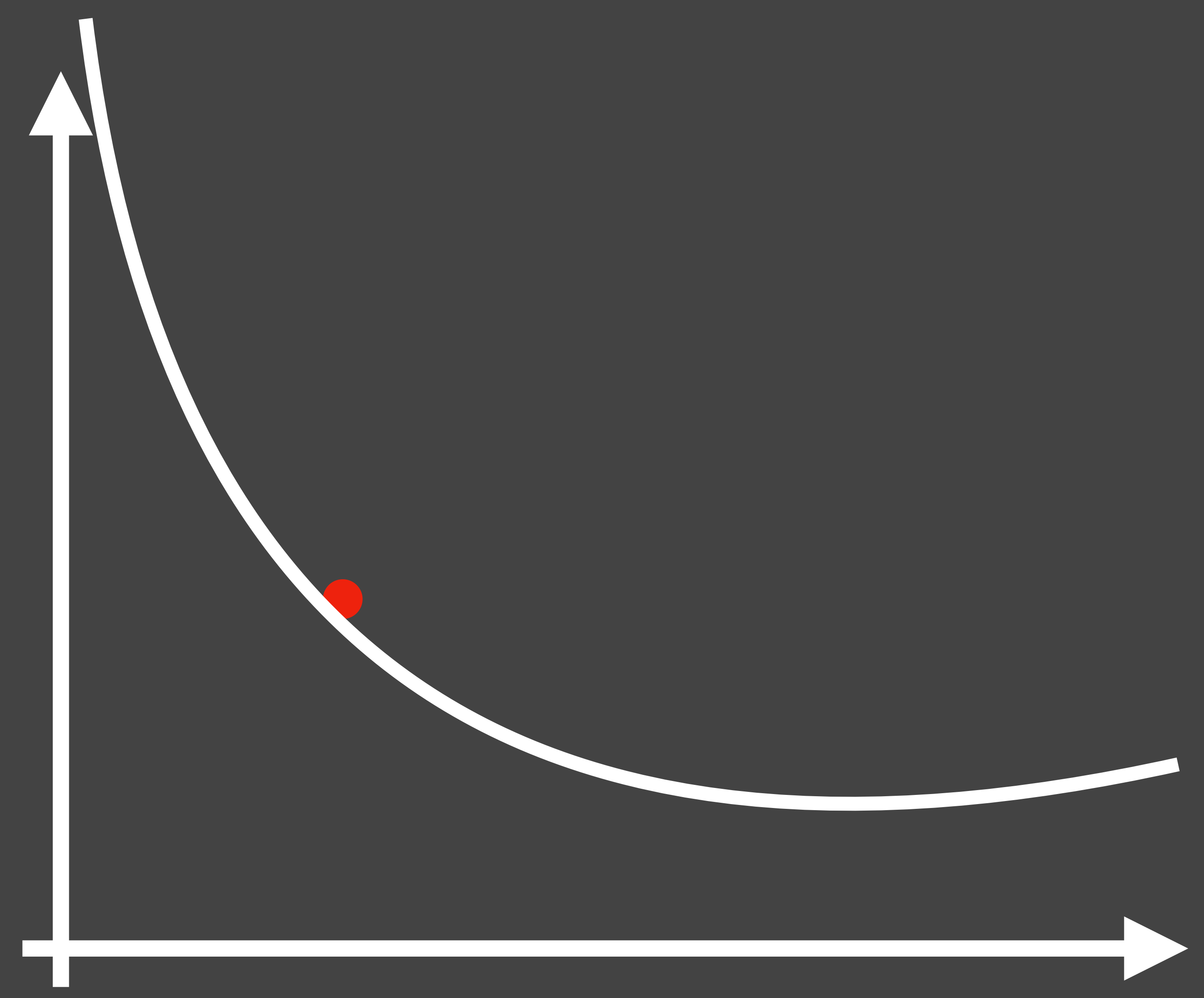
Dynamic

- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**

Comparison

Online

- Inserts Only
- Decisions are **irrevocable**



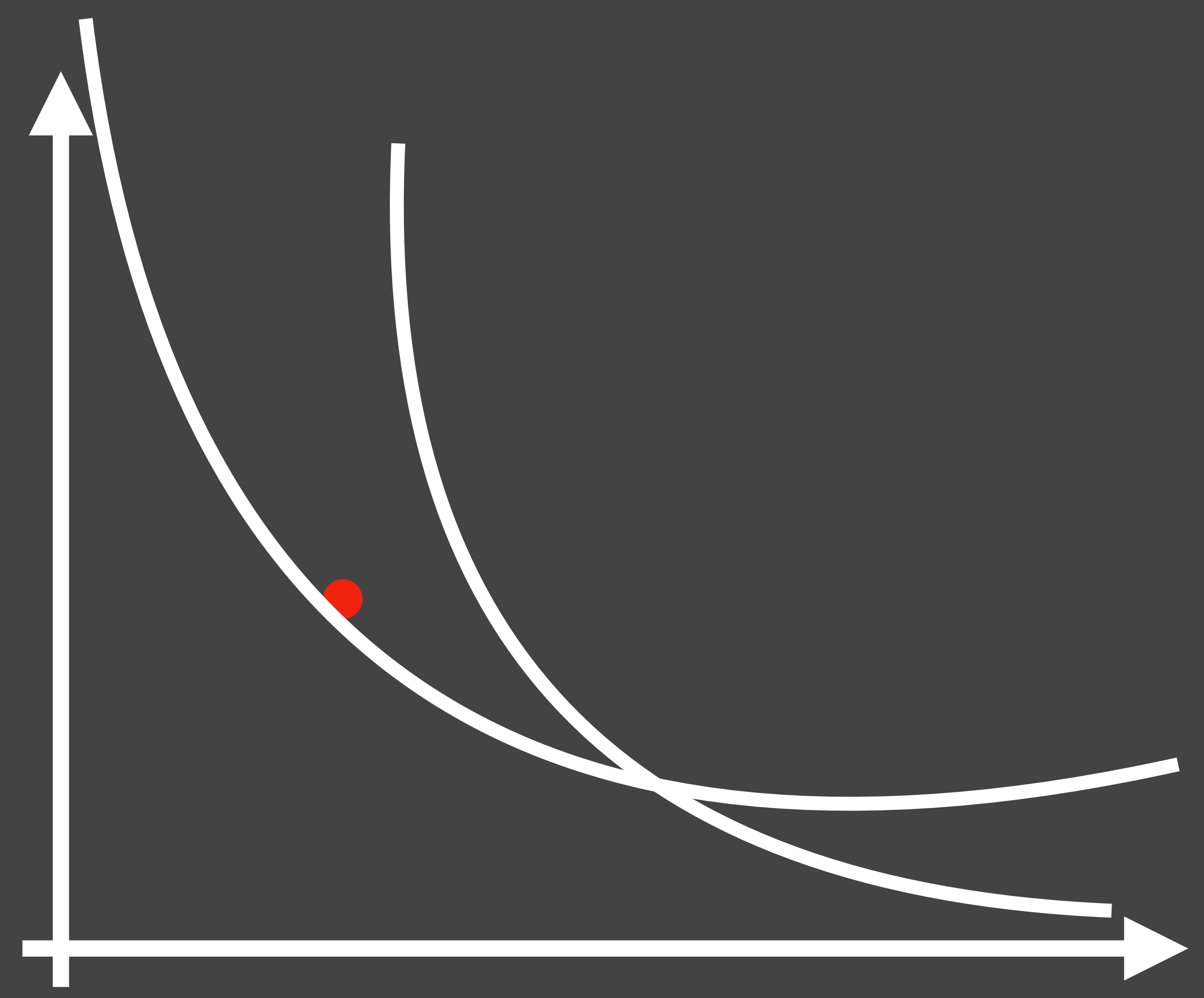
Dynamic

- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**

Comparison

Online

- Inserts Only
- Decisions are **irrevocable**



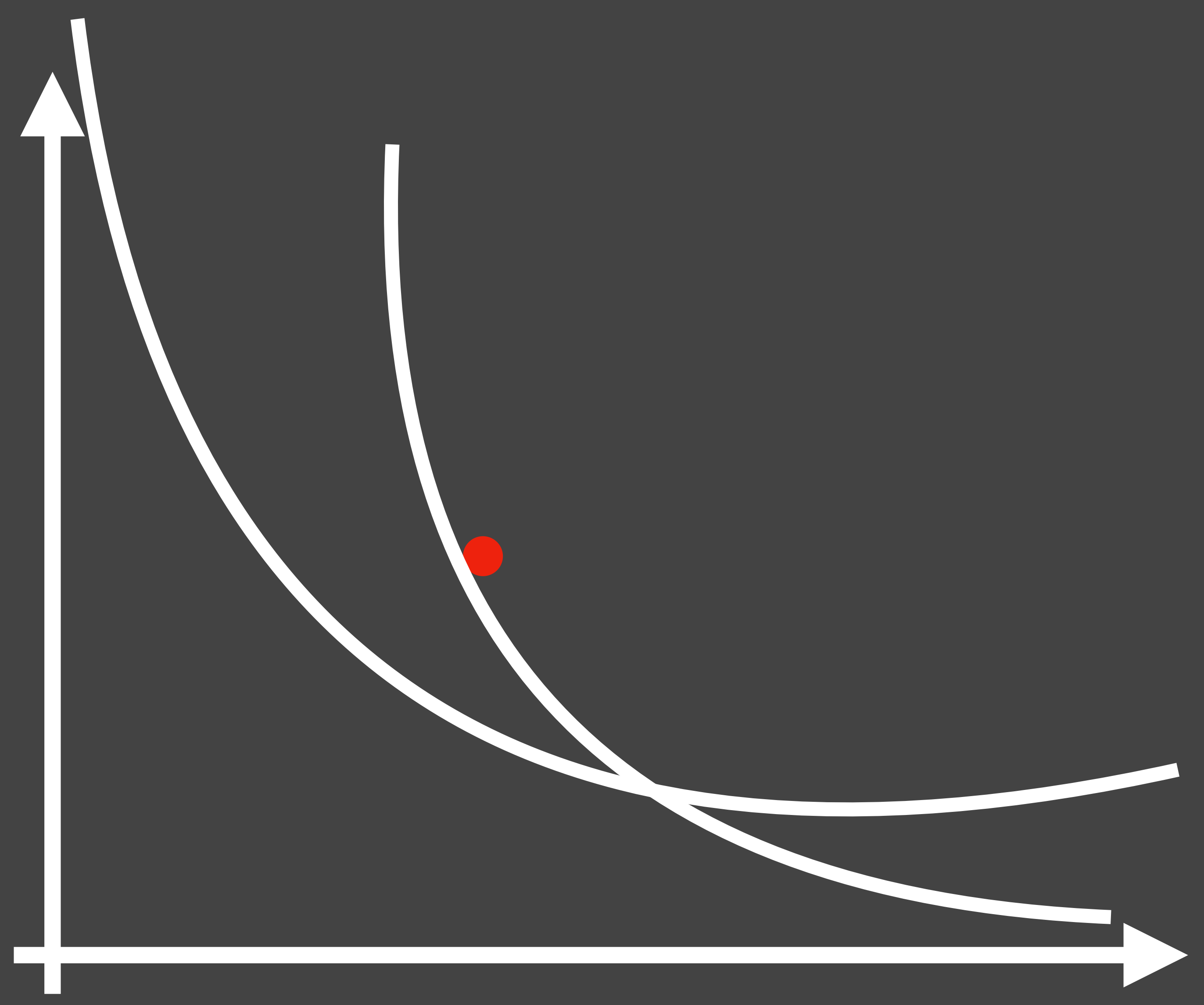
Dynamic

- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**

Comparison

Online

- Inserts Only
- Decisions are **irrevocable**



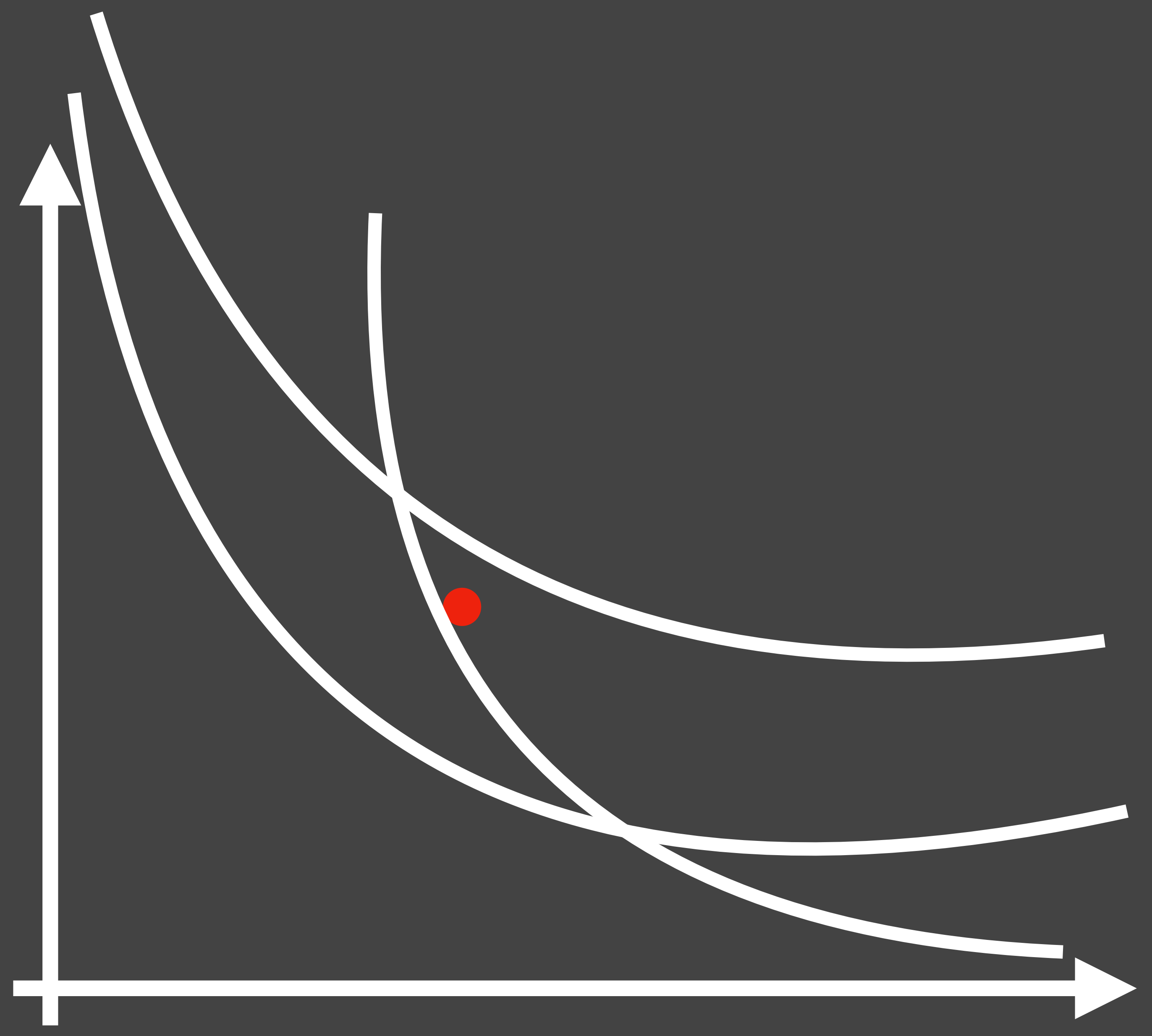
Dynamic

- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**

Comparison

Online

- Inserts Only
- Decisions are **irrevocable**



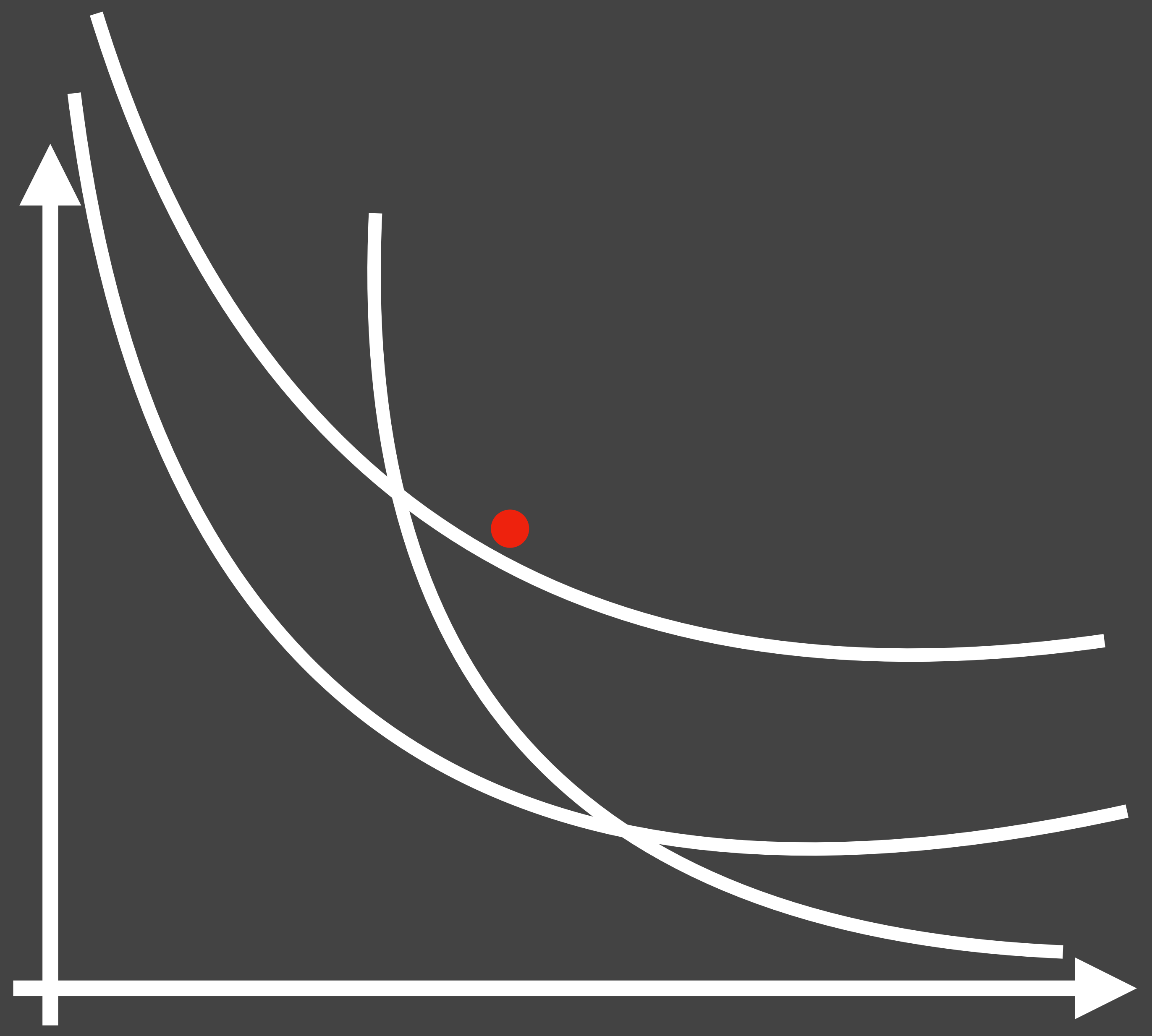
Dynamic

- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**

Comparison

Online

- Inserts Only
- Decisions are **irrevocable**



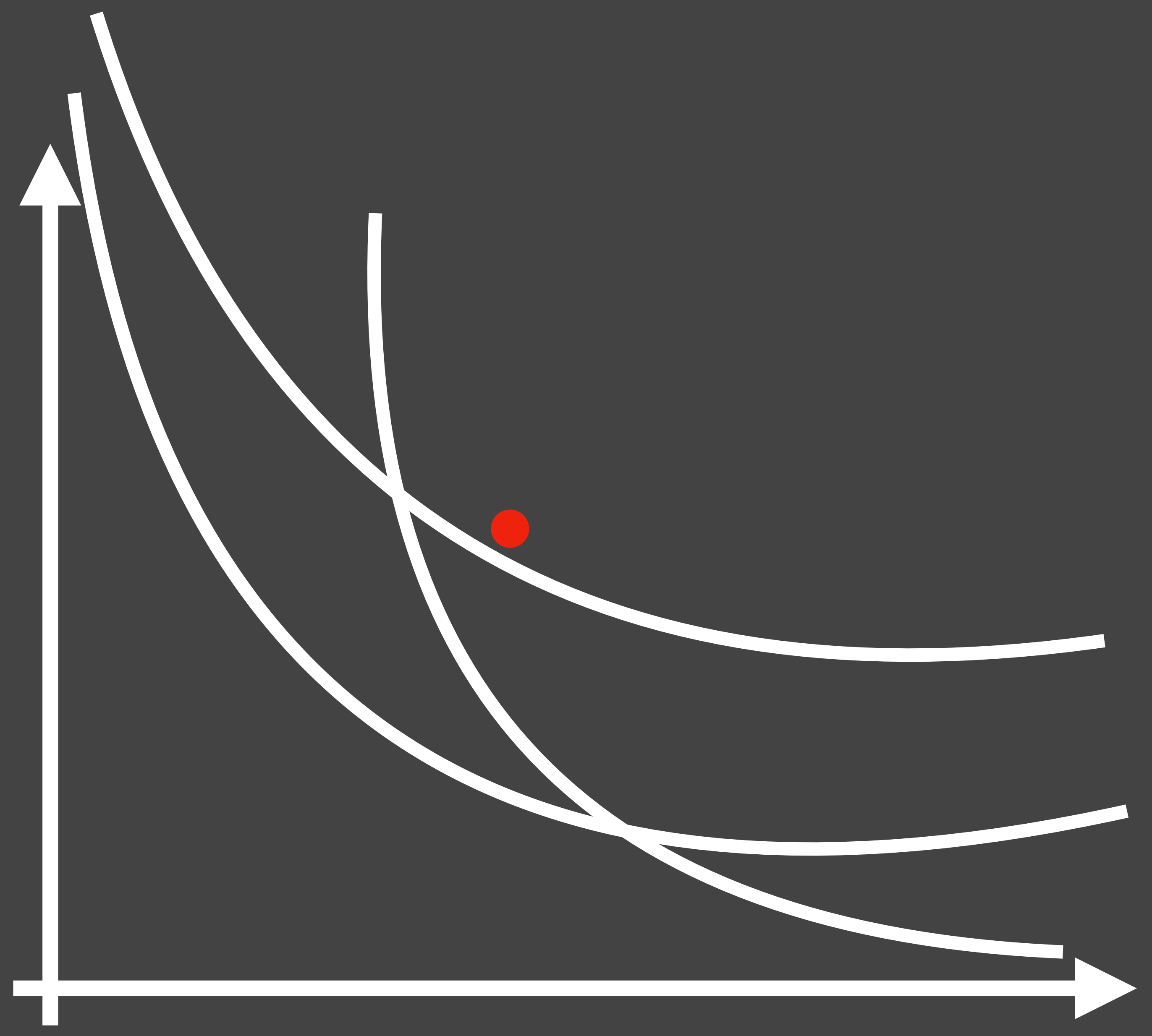
Dynamic

- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**

Comparison

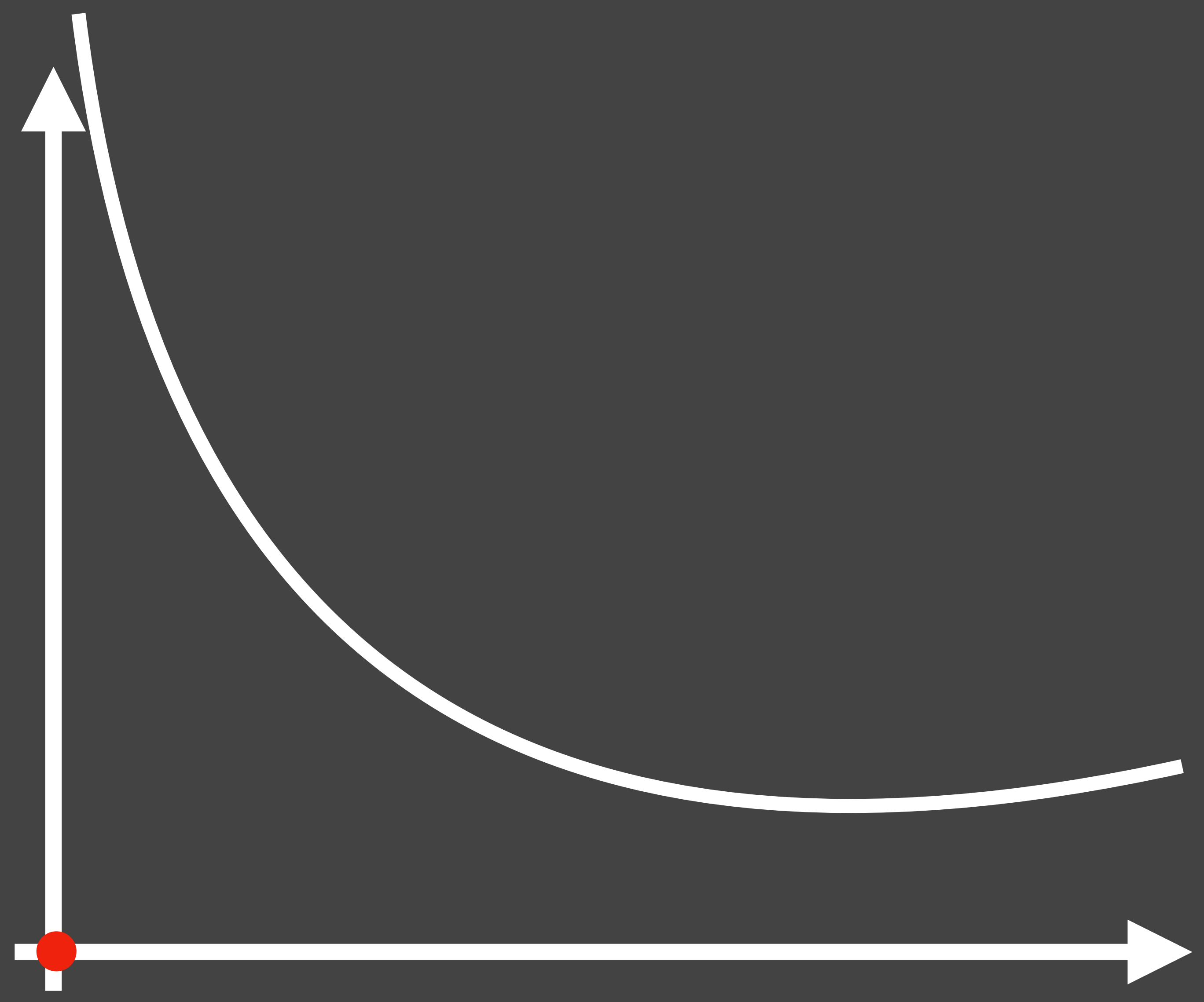
Online

- Inserts Only
- Decisions are **irrevocable**



Dynamic

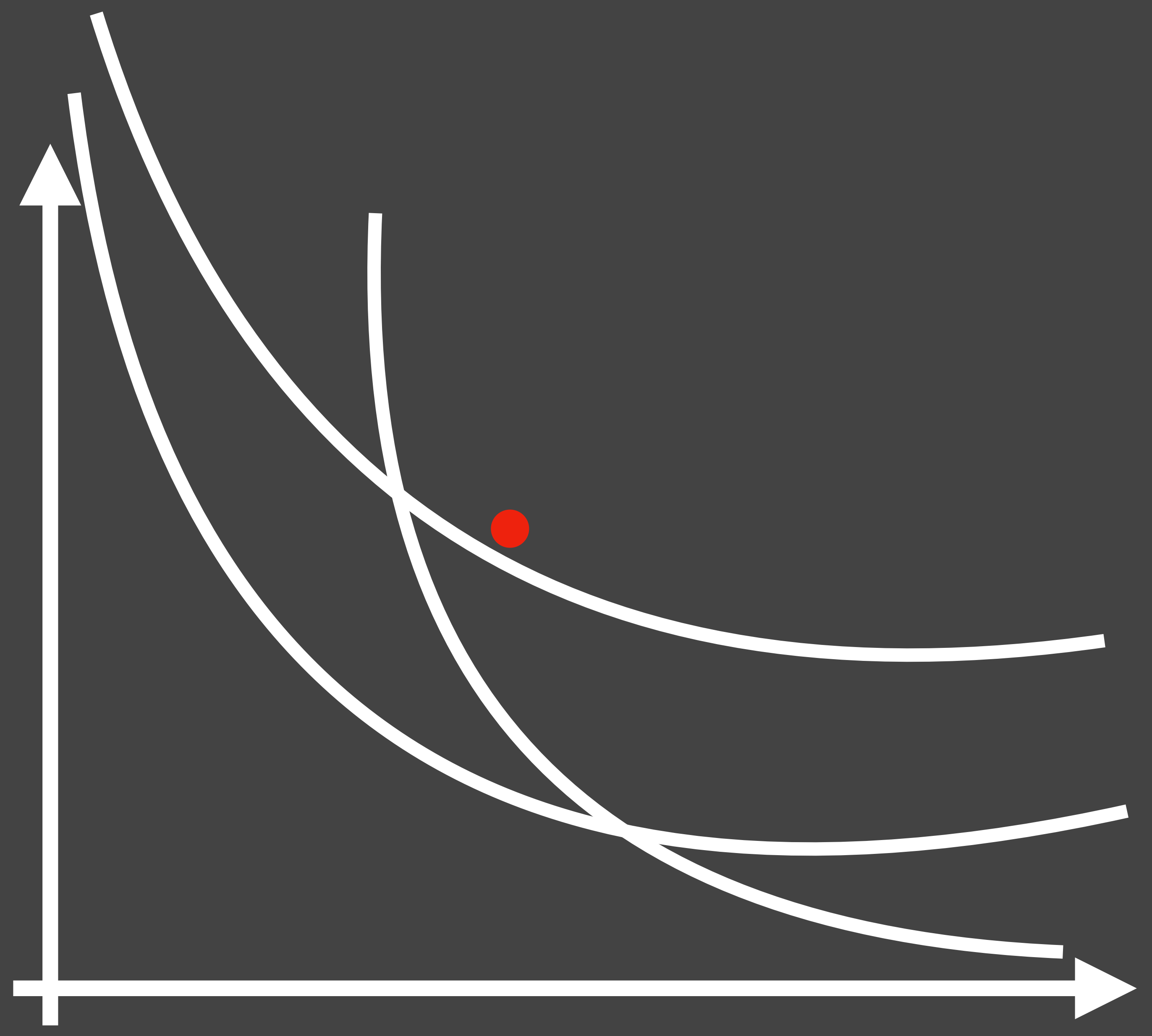
- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**



Comparison

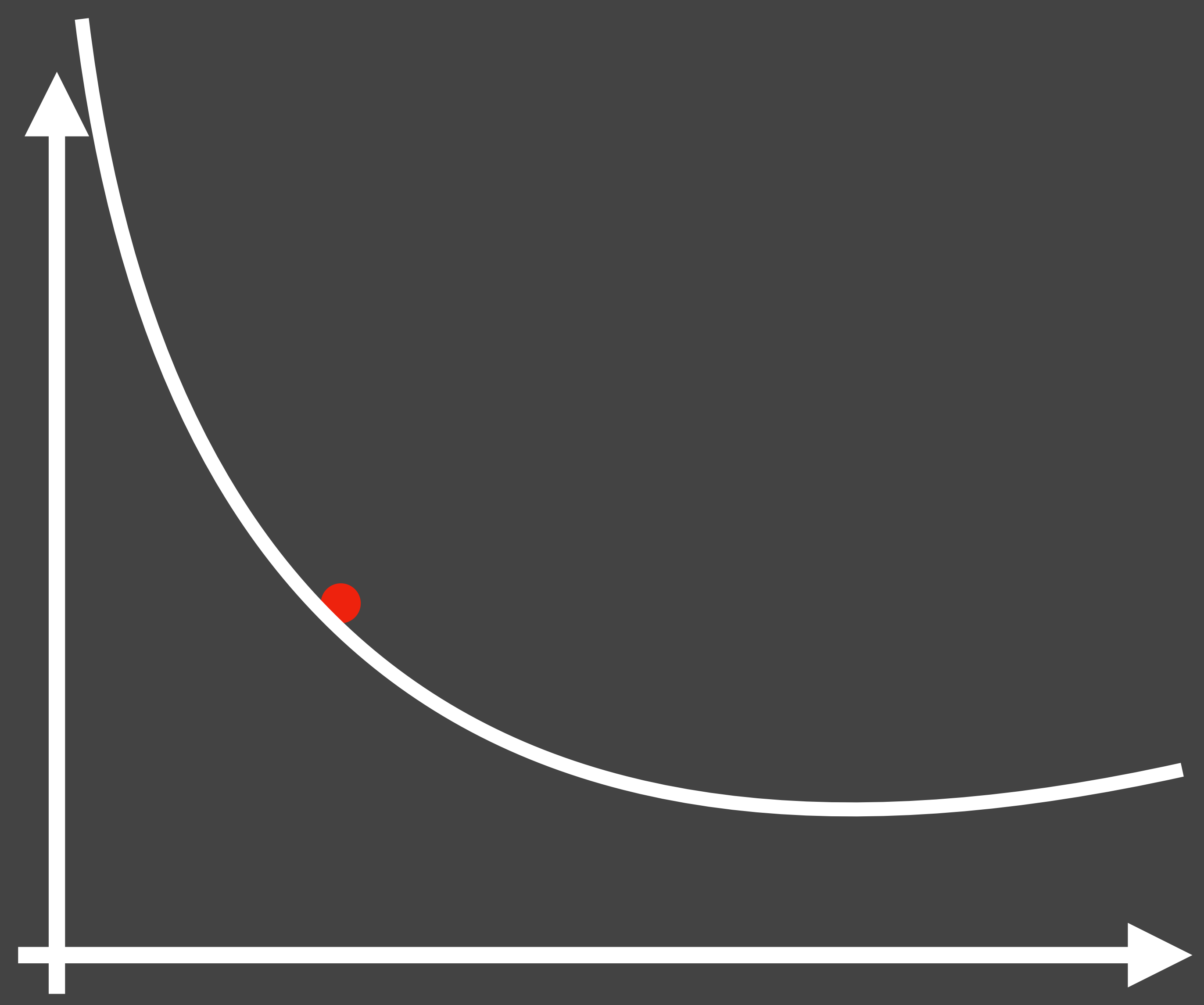
Online

- Inserts Only
- Decisions are **irrevocable**



Dynamic

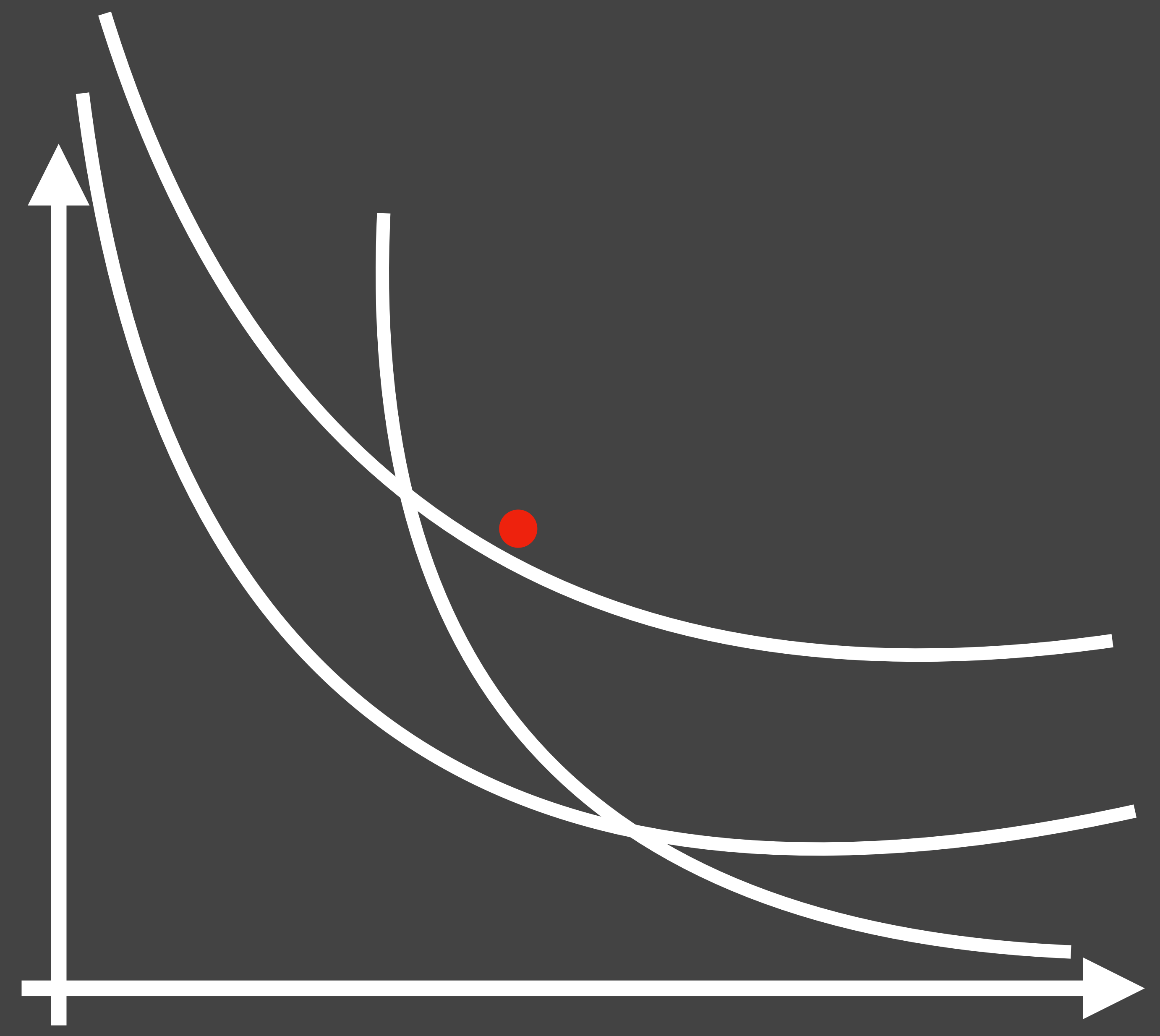
- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**



Comparison

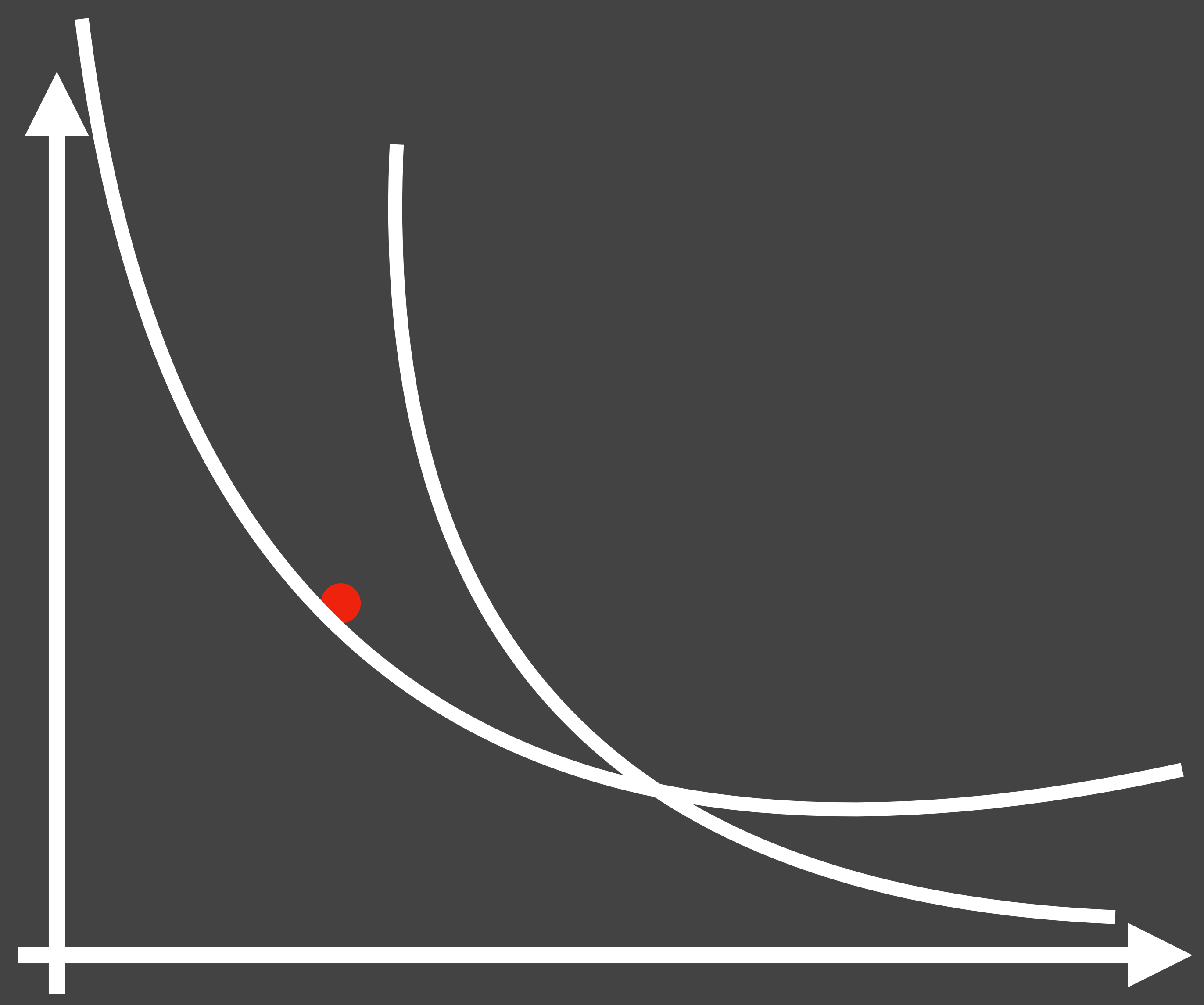
Online

- Inserts Only
- Decisions are **irrevocable**



Dynamic

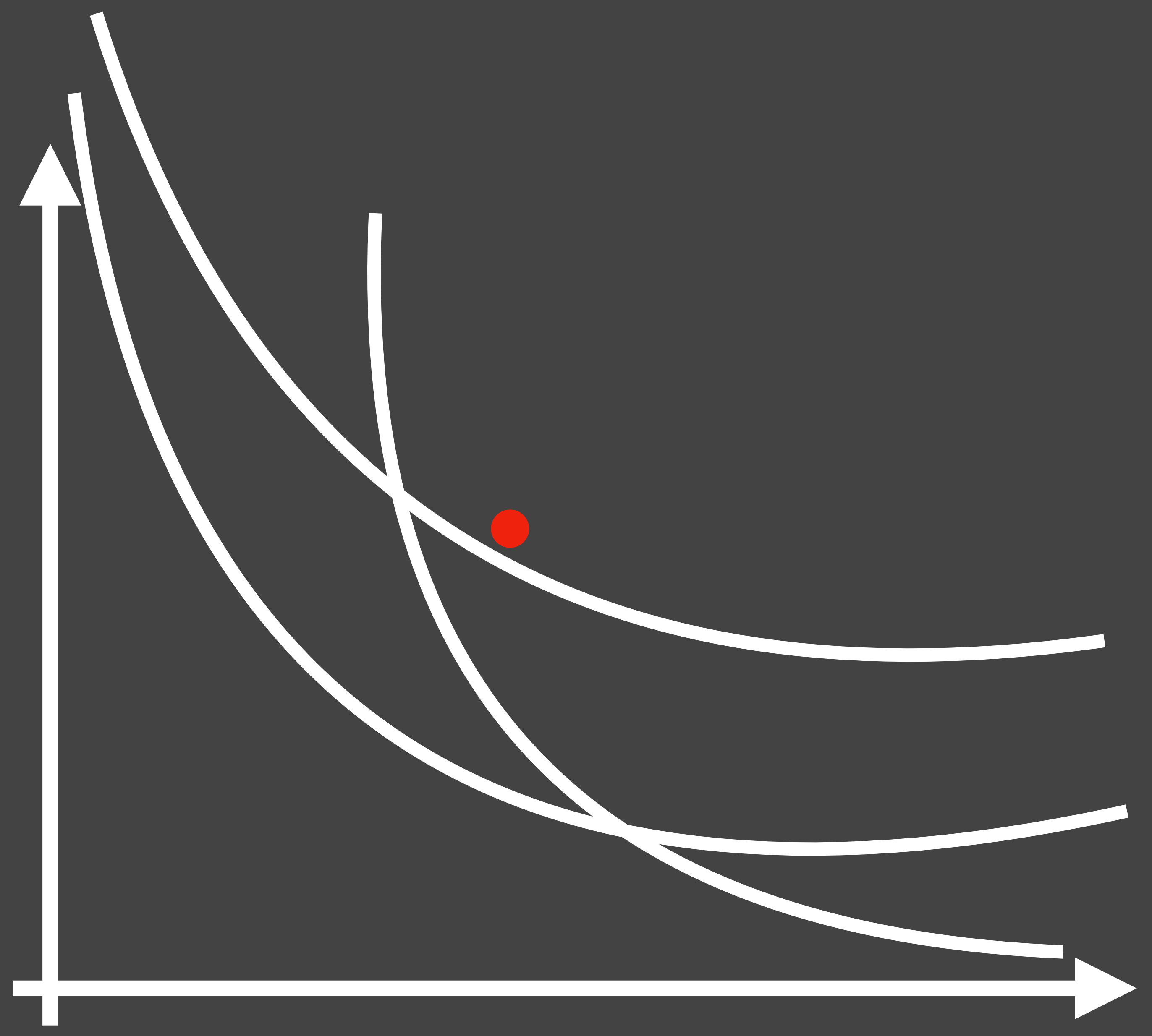
- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**



Comparison

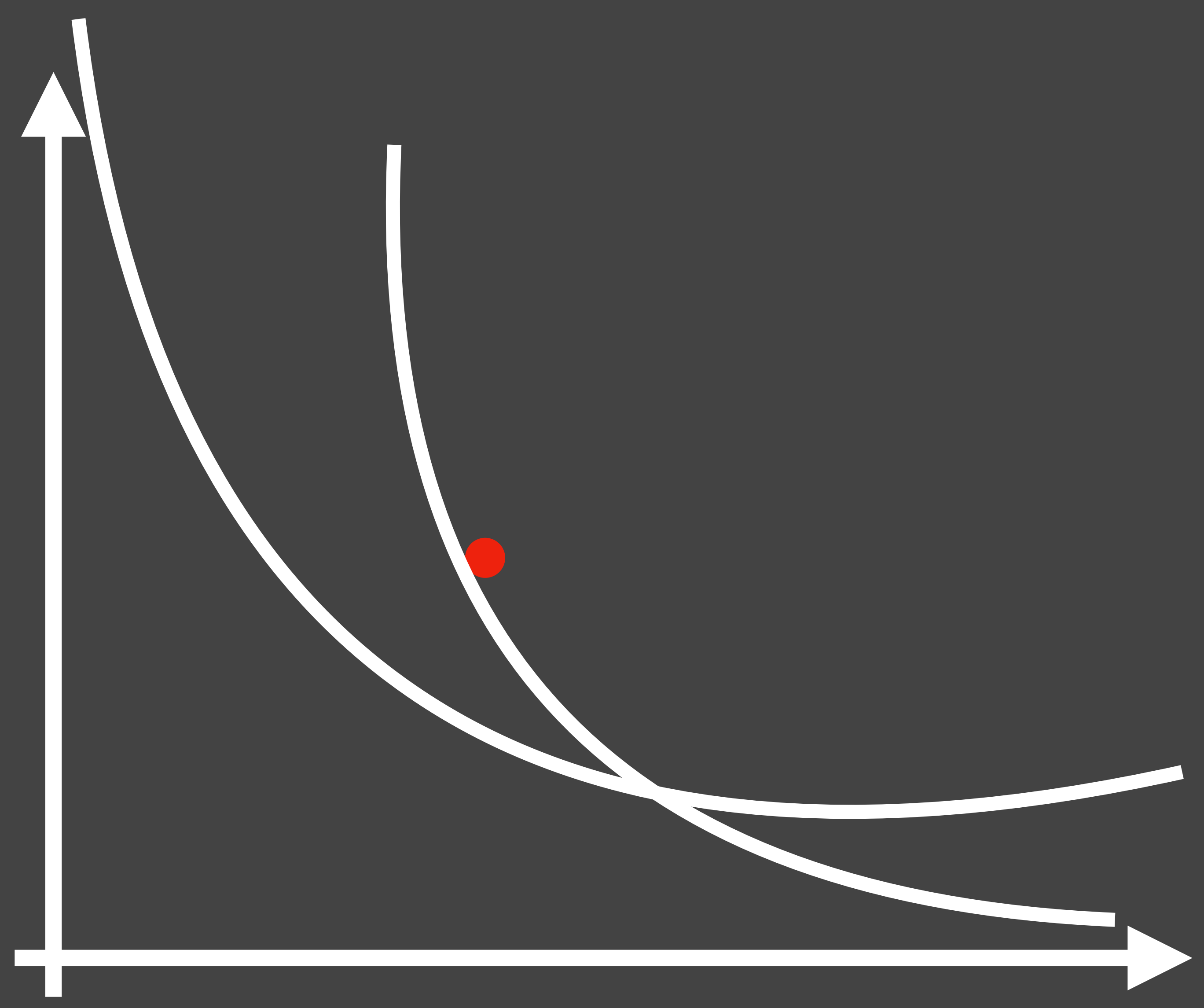
Online

- Inserts Only
- Decisions are **irrevocable**



Dynamic

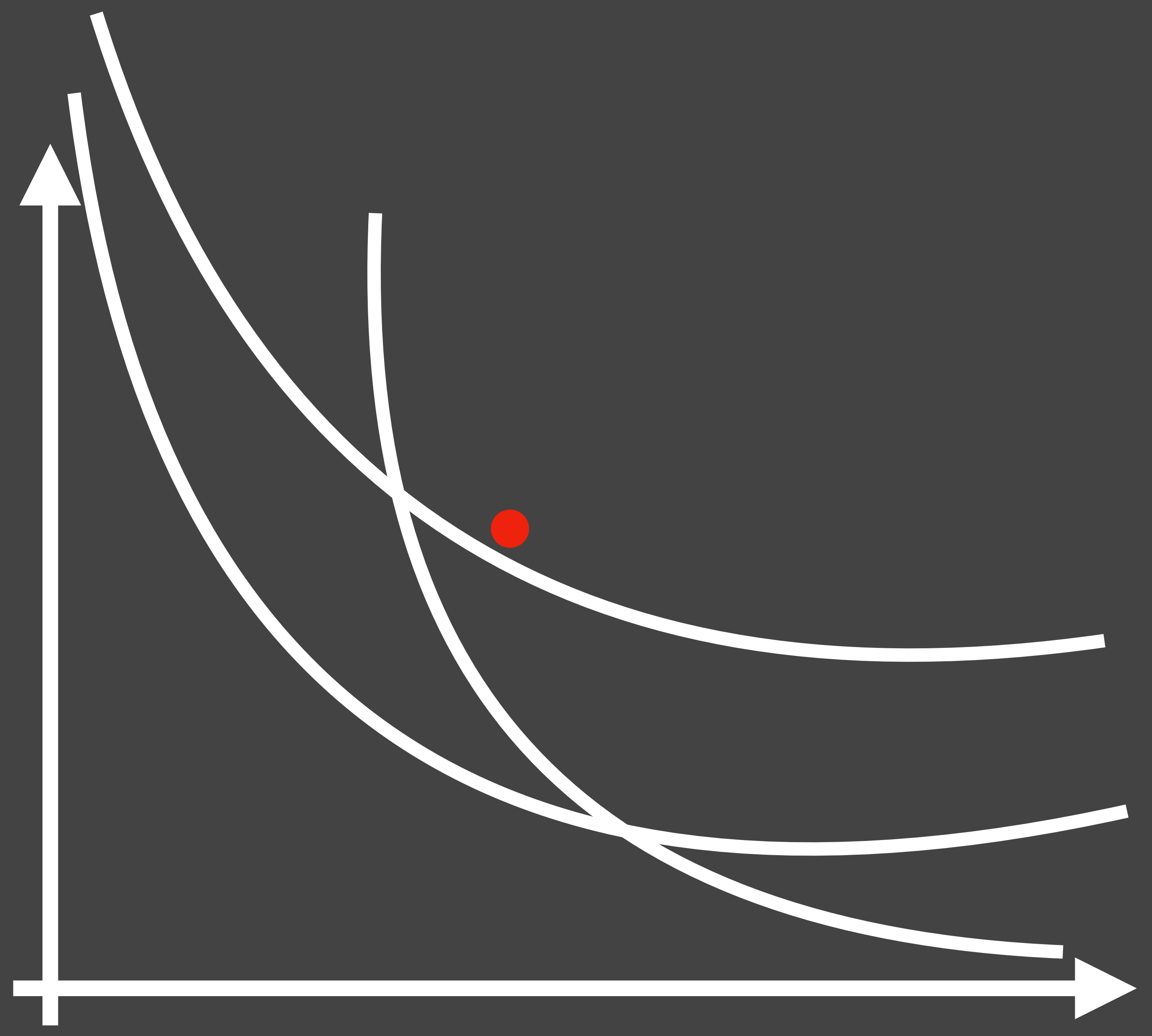
- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**



Comparison

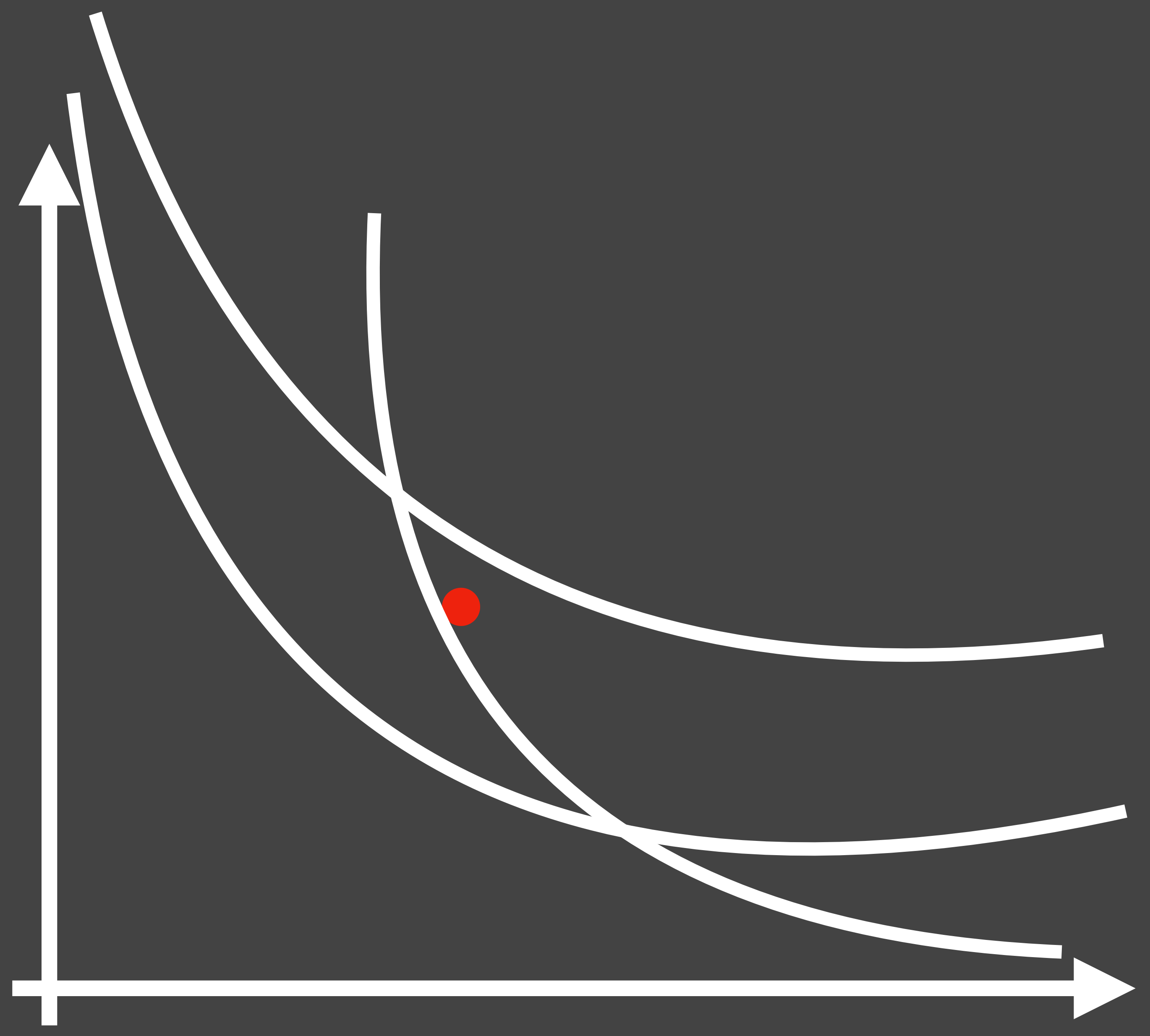
Online

- Inserts Only
- Decisions are **irrevocable**



Dynamic

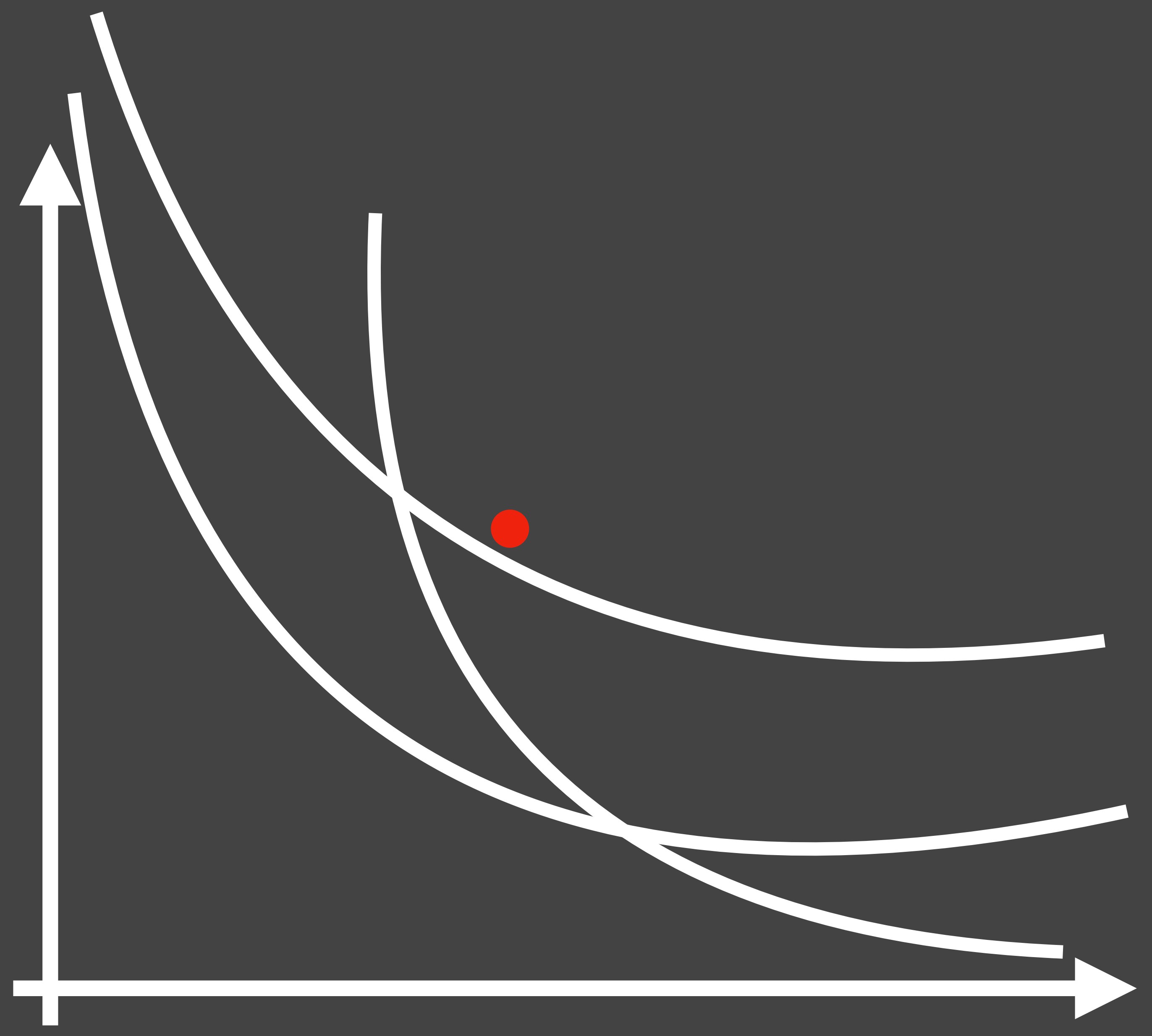
- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**



Comparison

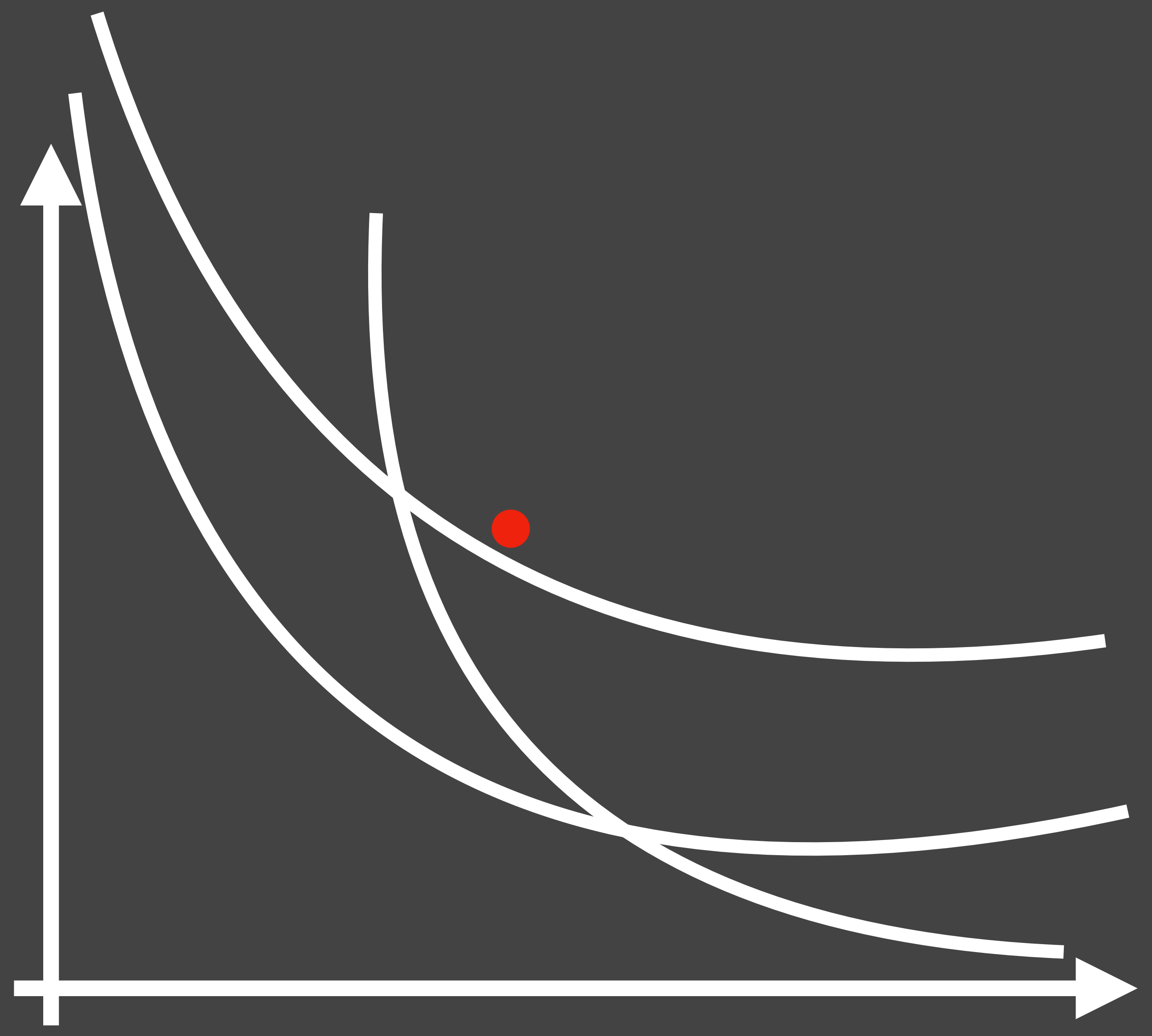
Online

- Inserts Only
- Decisions are **irrevocable**



Dynamic

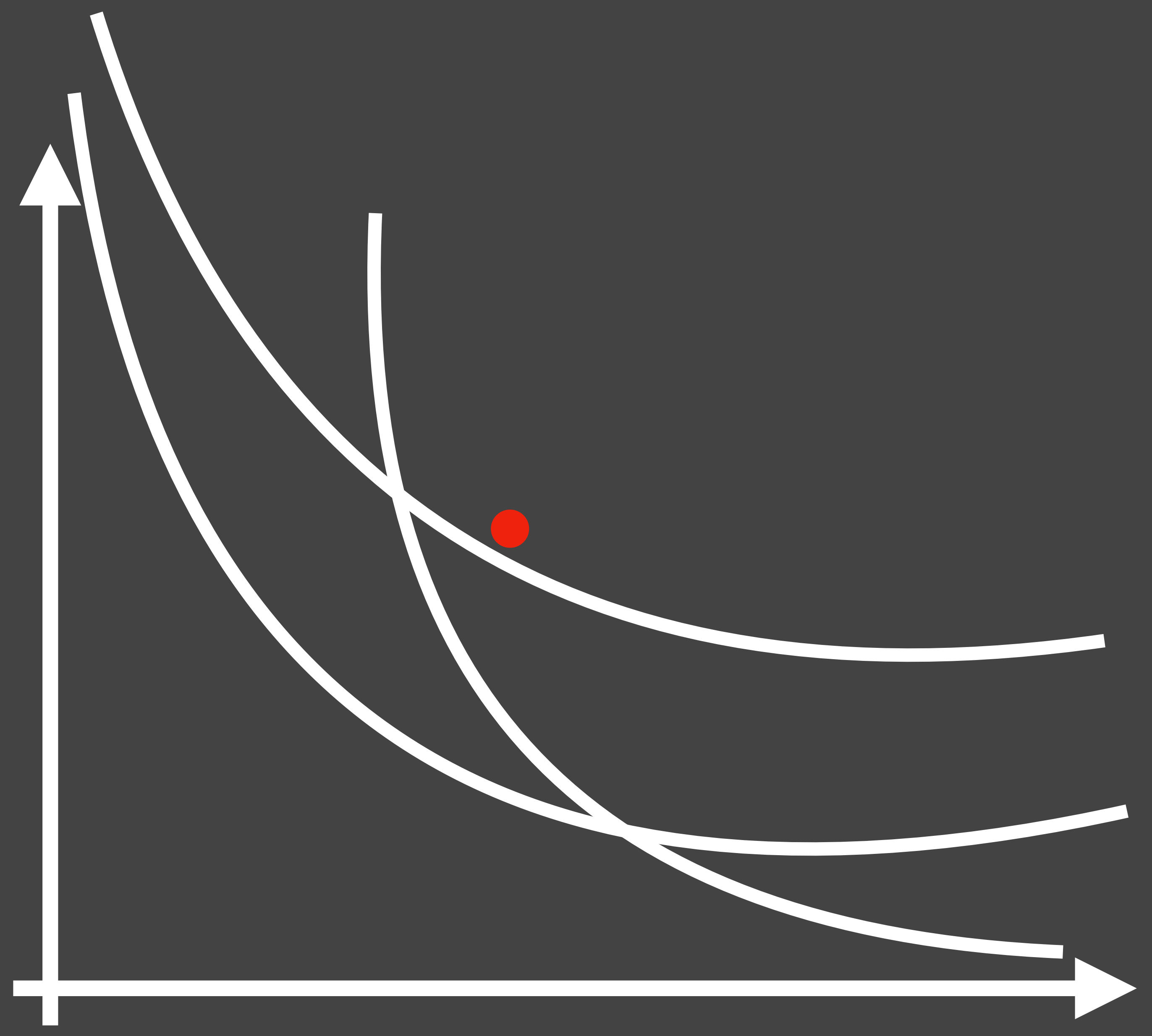
- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**



Comparison

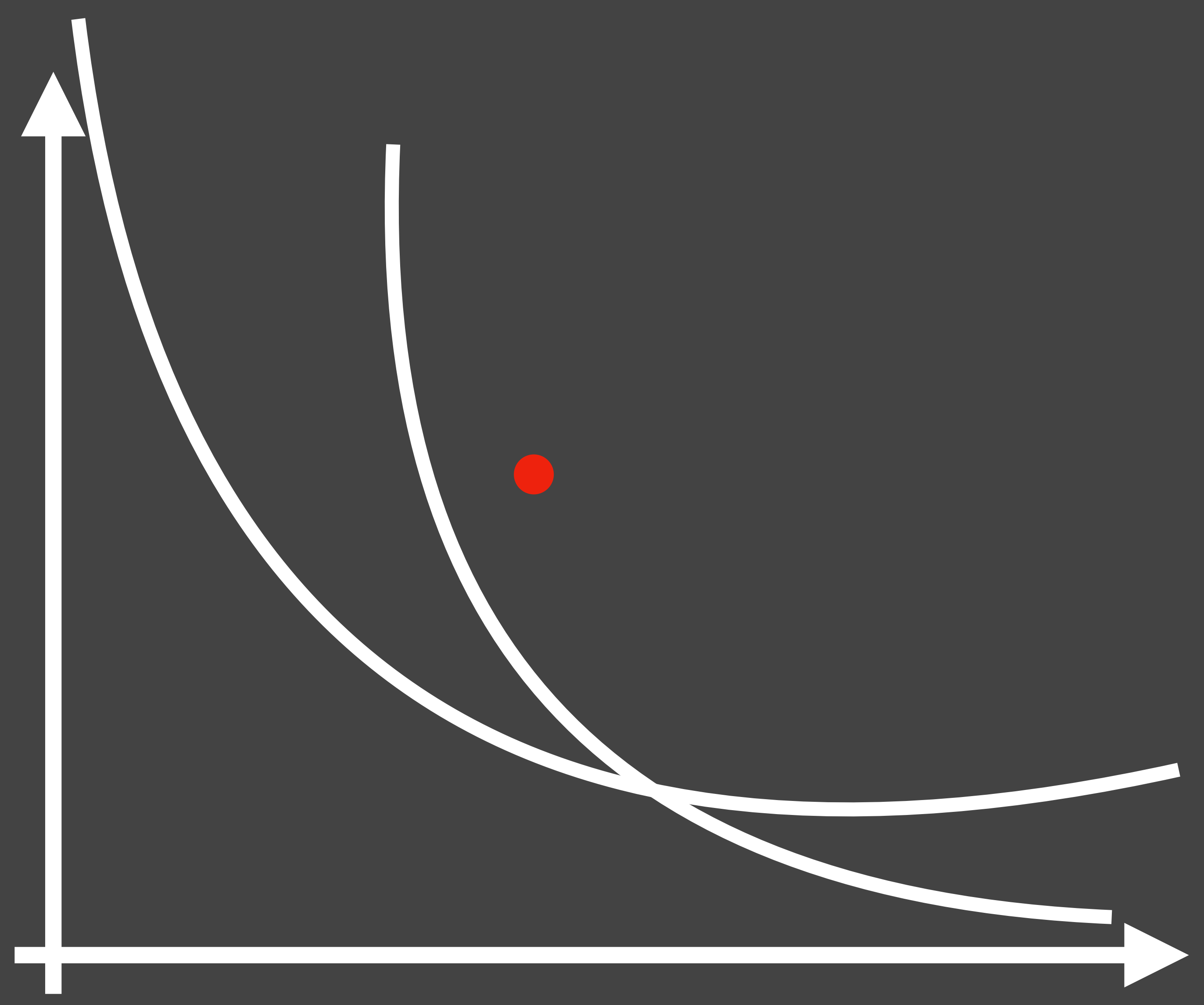
Online

- Inserts Only
- Decisions are **irrevocable**



Dynamic

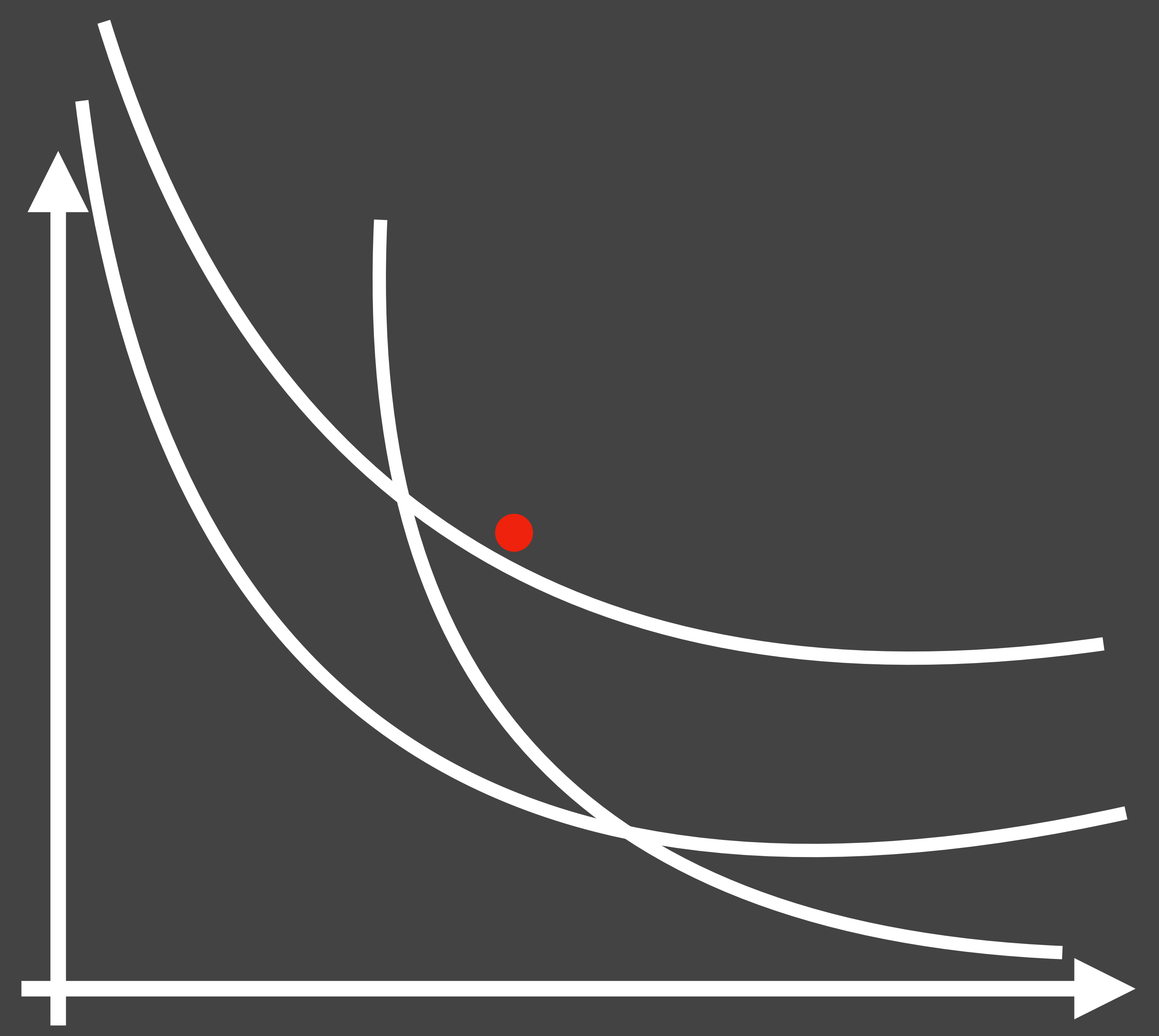
- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**



Comparison

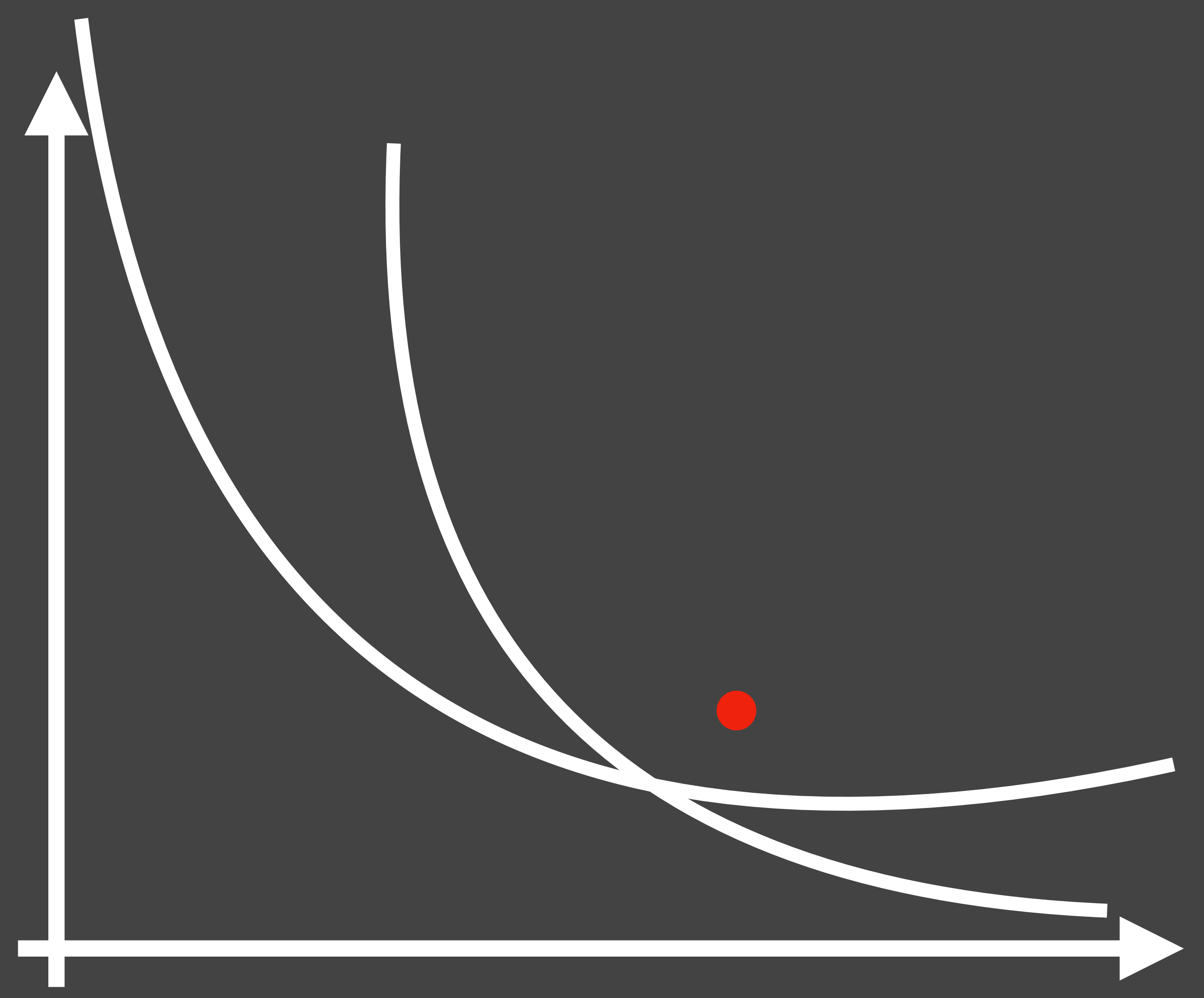
Online

- Inserts Only
- Decisions are **irrevocable**



Dynamic

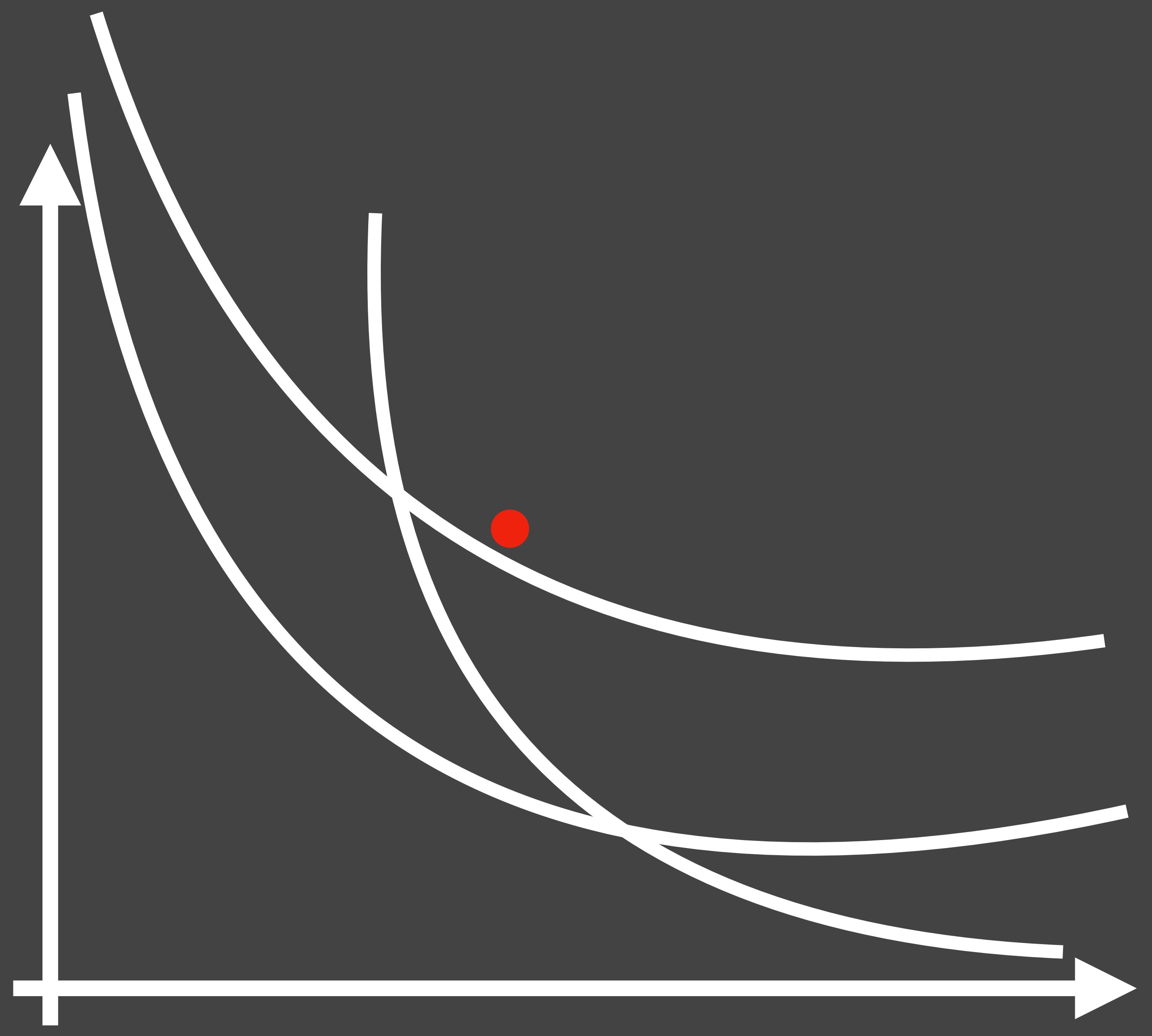
- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**



Comparison

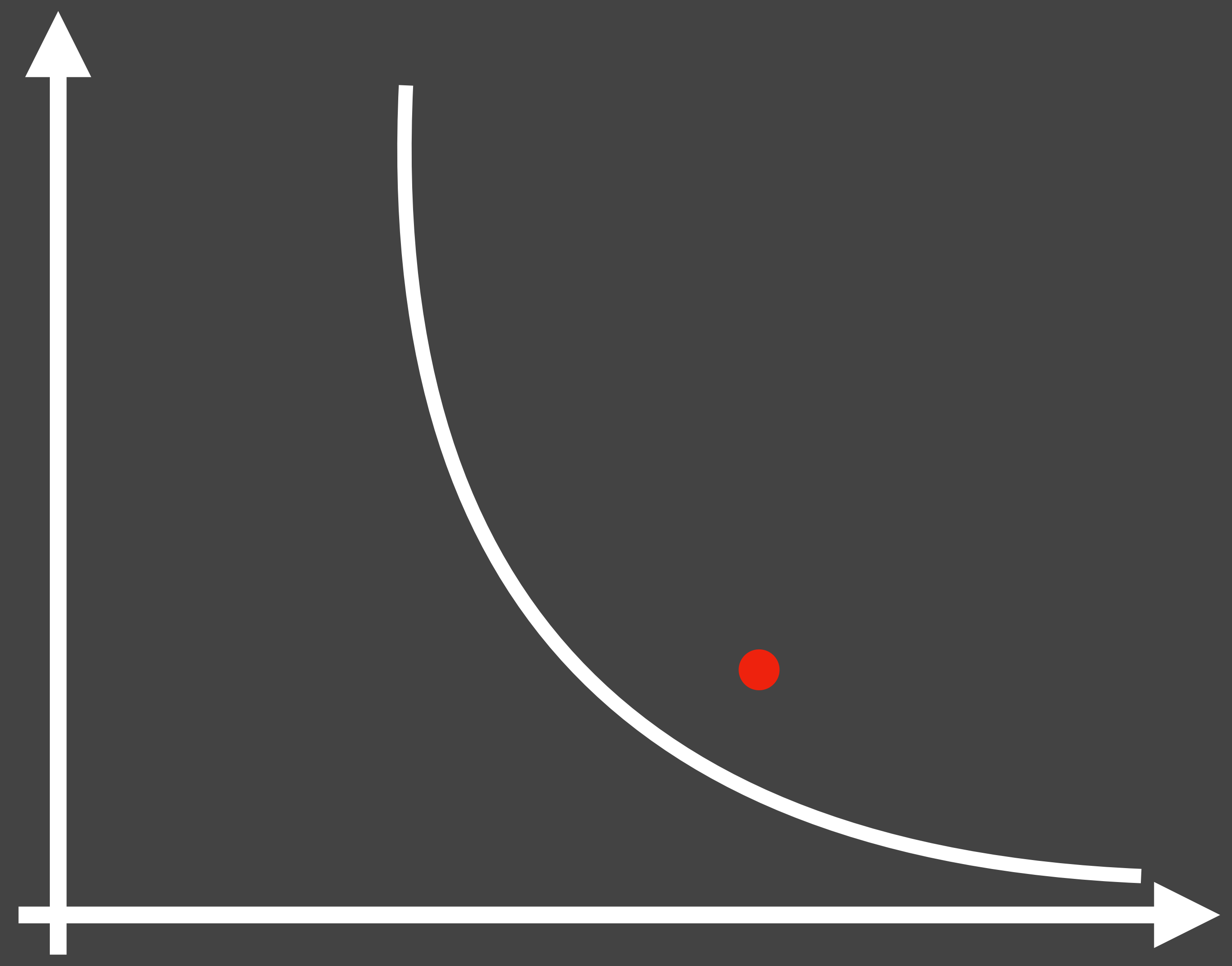
Online

- Inserts Only
- Decisions are **irrevocable**



Dynamic

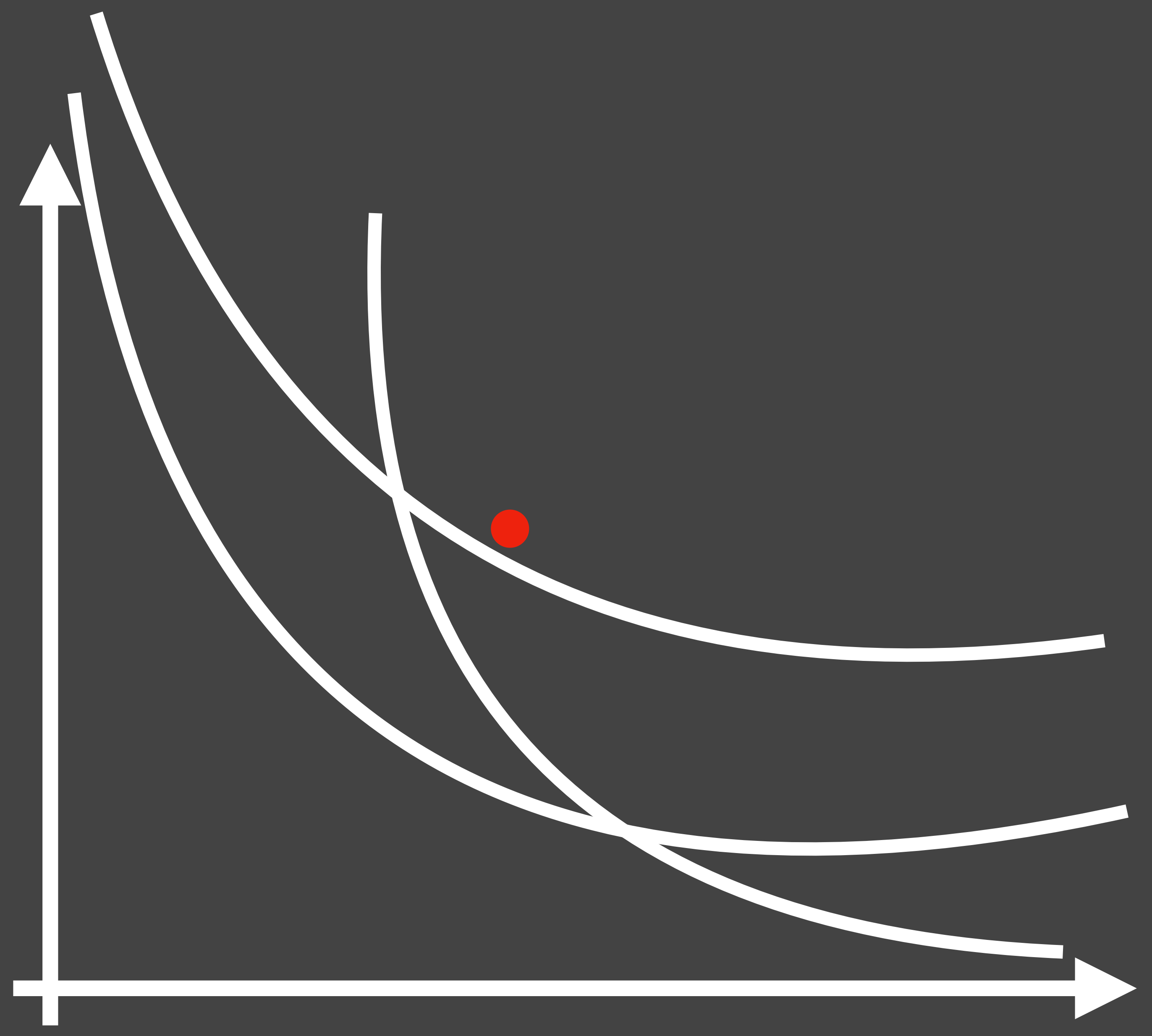
- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**



Comparison

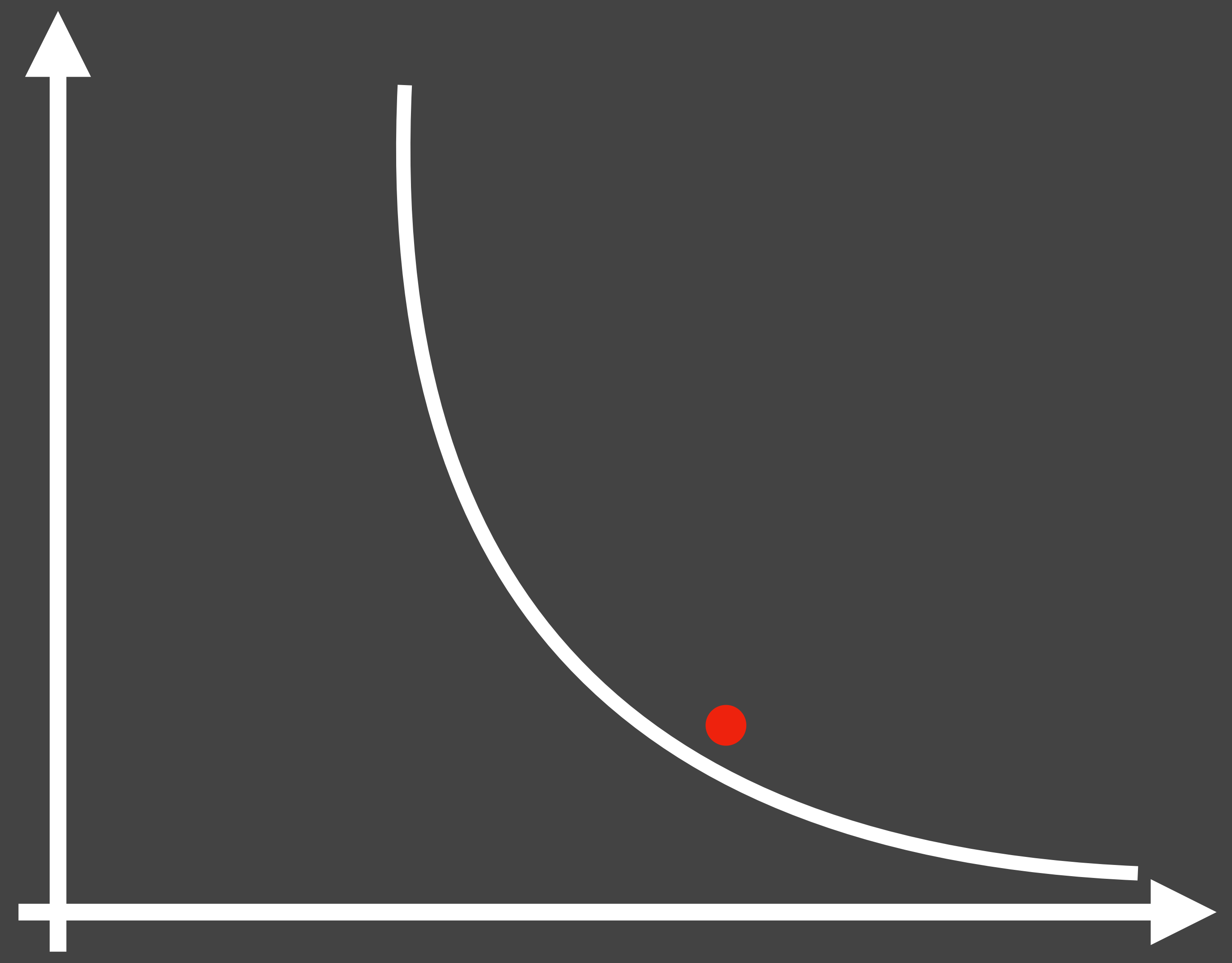
Online

- Inserts Only
- Decisions are **irrevocable**



Dynamic

- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**



Comparison

Online

- Inserts Only
- Decisions are **irrevocable**

Theorem (Online) [Gupta L. SODA 20]:

Approximation $O(\log^2 n)$.

Dynamic

- Inserts + Deletes
- Want minimum # edits, a.k.a. **recourse.**

Theorem (Dynamic) [Gupta L. FOCS 20]:

(i) **Approximation** $O(\log n)$.

(ii) **Recourse** $\tilde{O}(1)$.

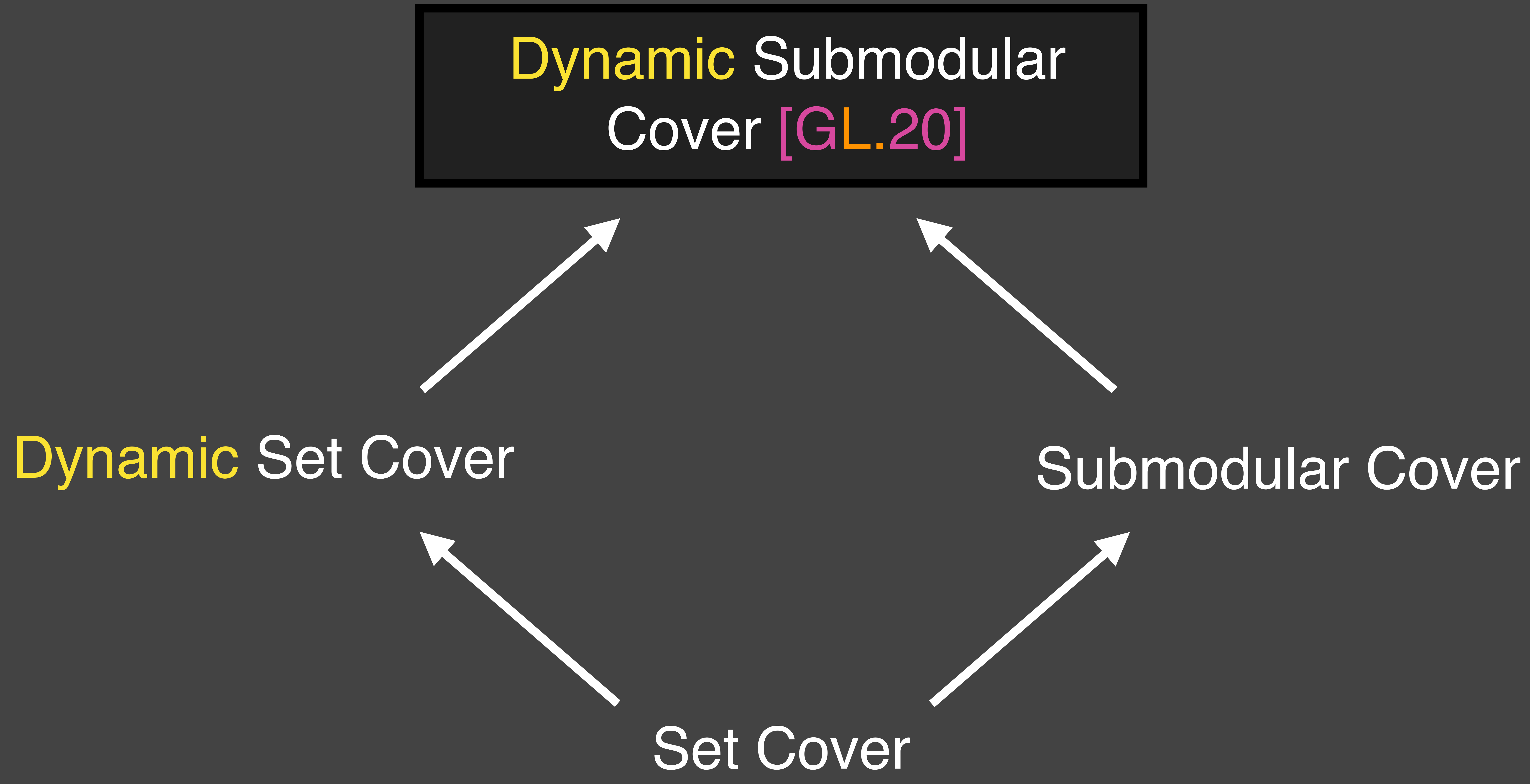
Dynamic Submodular Cover [Gupta L. FOCS 20]

Dynamic Set Cover

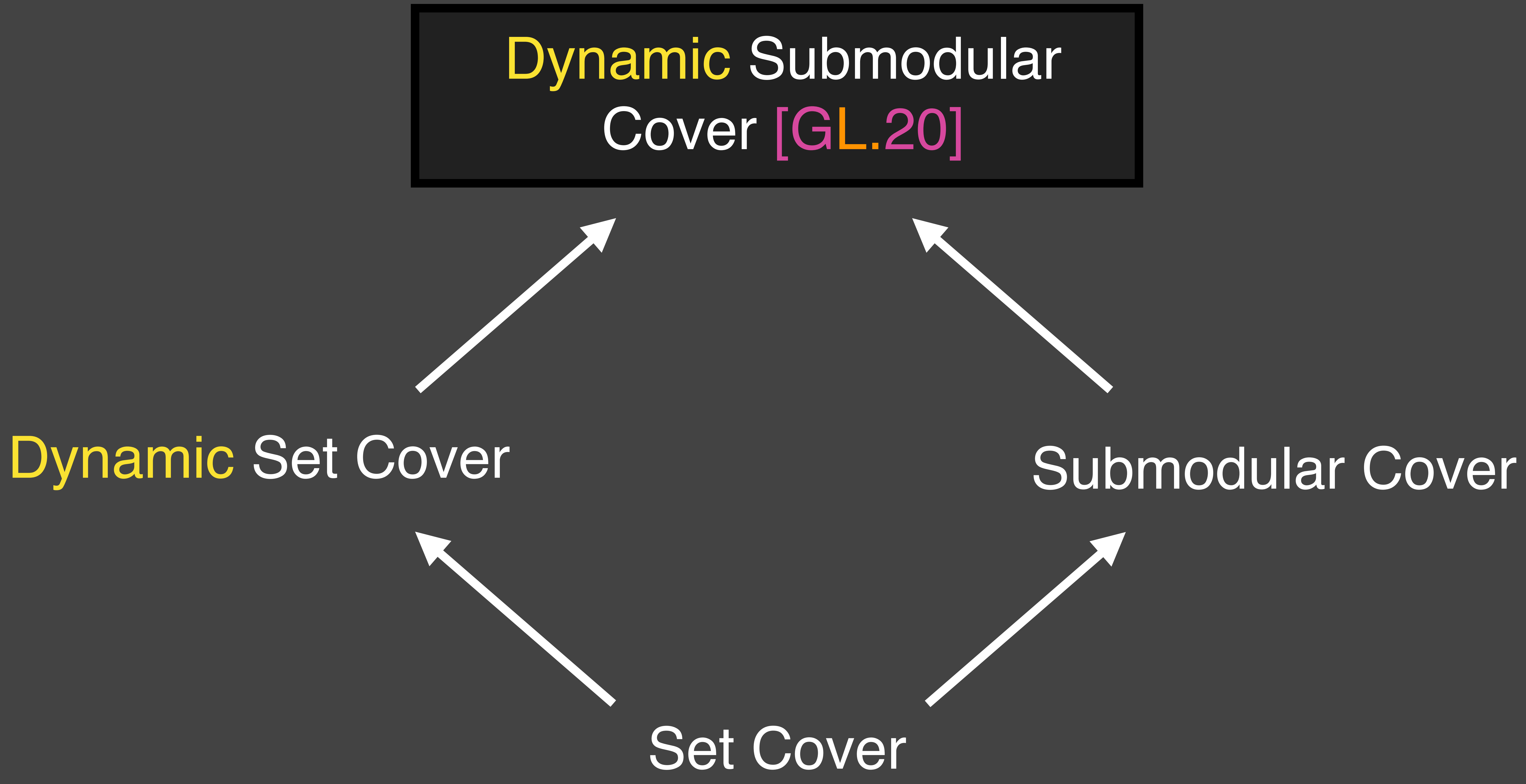
Submodular Cover

Set Cover

Dynamic Submodular Cover [Gupta L. FOCS 20]



Dynamic Submodular Cover [Gupta L. FOCS 20]



Modeling power of Submodular Cover + Dynamic.

Is There a Theory to Build?

Is There a Theory to Build?

Most work (mine included!) based on 1-off combinatorial insights.

Is There a Theory to Build?

Most work (mine included!) based on 1-off combinatorial insights.

- Difficult to come up with. 😰

Is There a Theory to Build?

Most work (mine included!) based on 1-off combinatorial insights.

- Difficult to come up with. 😰
- Difficult to generalize. 😰

Is There a Theory to Build?

Most work (mine included!) based on 1-off combinatorial insights.

- Difficult to come up with. 😰
- Difficult to generalize. 😰

General recipe for designing stable algorithms?

Is There a Theory to Build?

(Yes)

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

Is There a Theory to Build?

(Yes)

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

<u>Theorem</u> [Bhattacharya, Buchbinder, L., Saranurak, In submission]:

Dynamic Linear Programming with movement $O(\log n) \cdot \mathrm{OPT}$.

Is There a Theory to Build?

(Yes)

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

<u>Theorem</u> [Bhattacharya, Buchbinder, L., Saranurak, In submission]:

Dynamic Linear Programming with movement $O(\log n) \cdot$ OPT.

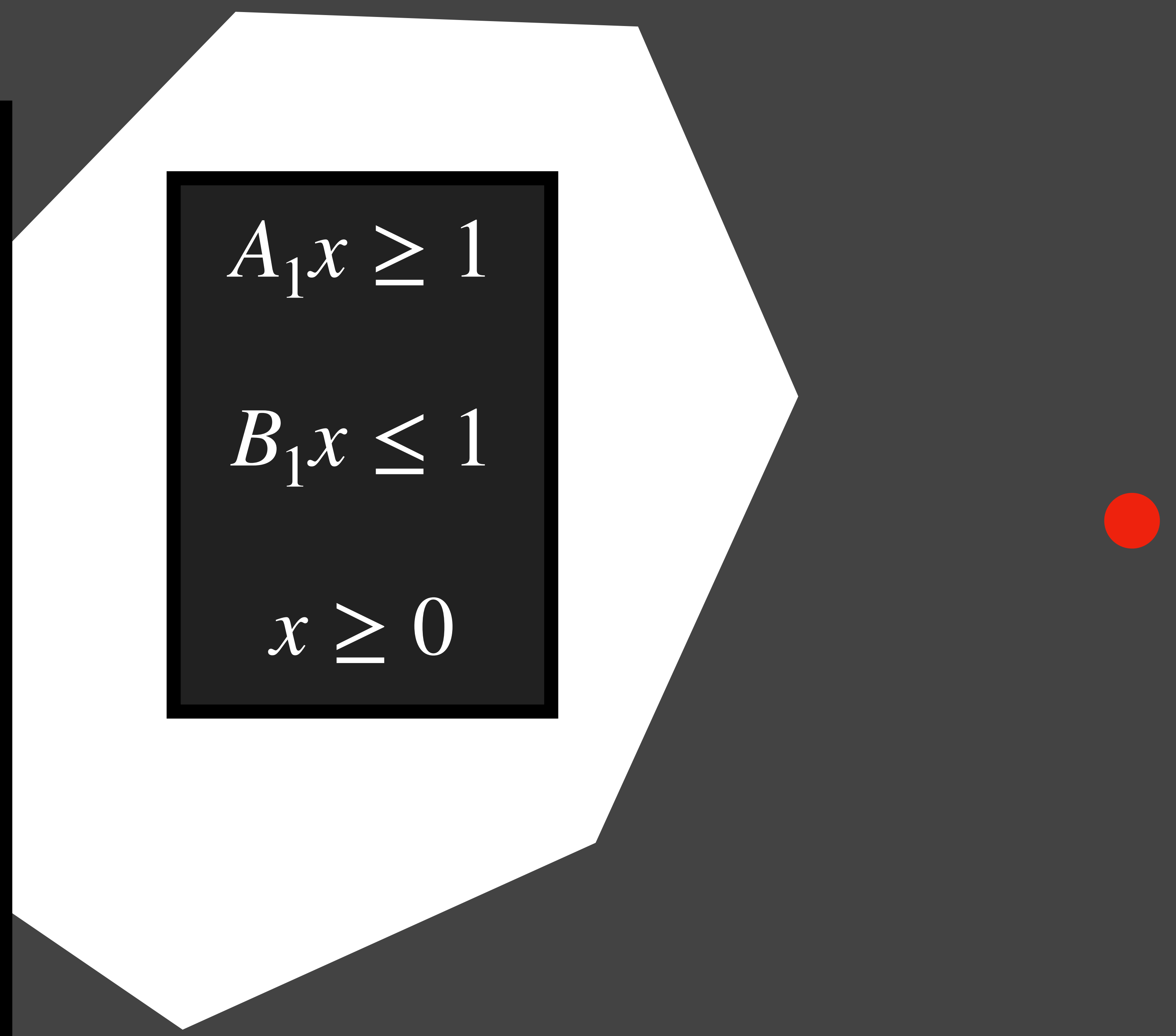
Is There a Theory to Build?

(Yes)

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

Theorem [Bhattacharya, Buchbinder, L., Saranurak, In submission]:

Dynamic Linear Programming with movement $O(\log n) \cdot \mathrm{OPT}$.



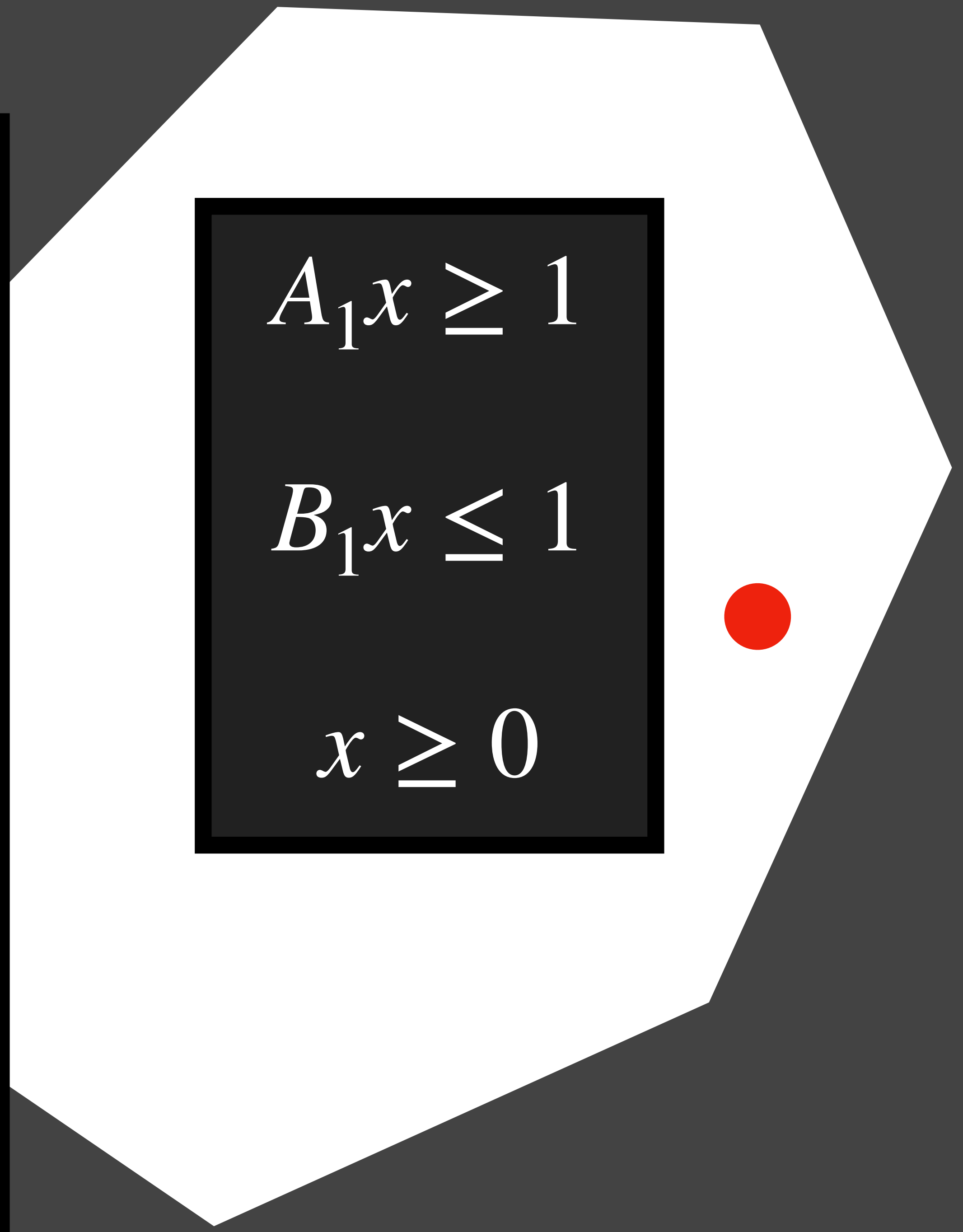
Is There a Theory to Build?

(Yes)

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

<u>Theorem</u> [Bhattacharya, Buchbinder, L., Saranurak, In submission]:

Dynamic Linear Programming with movement $O(\log n) \cdot$ OPT.

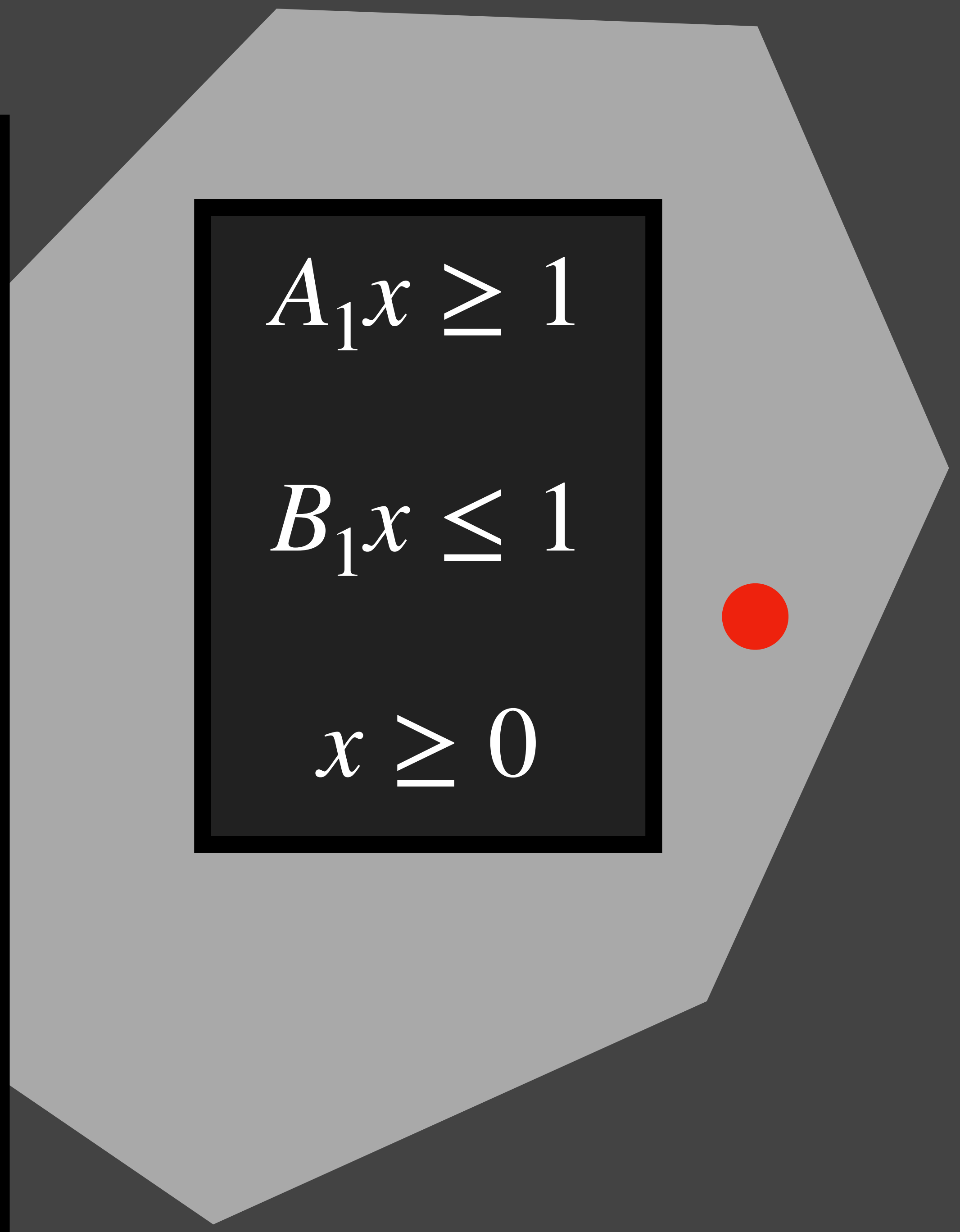


Is There a Theory to Build?

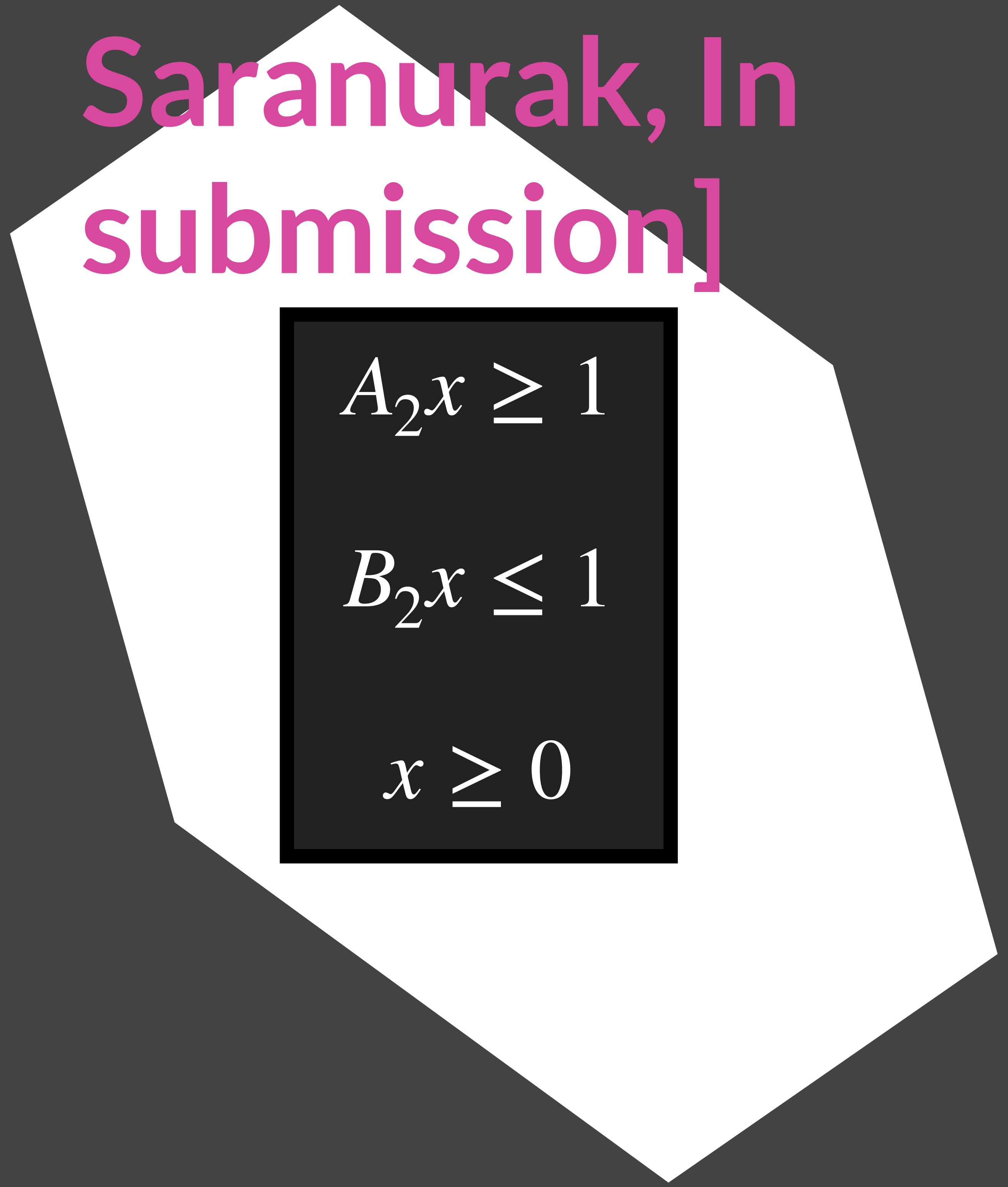
(Yes)

Theorem [Bhattacharya, Buchbinder, L., Saranurak, In submission]:

Dynamic Linear Programming with movement $O(\log n) \cdot \mathrm{OPT}$.



[Bhattacharya, Buchbinder, L., Saranurak, In submission]

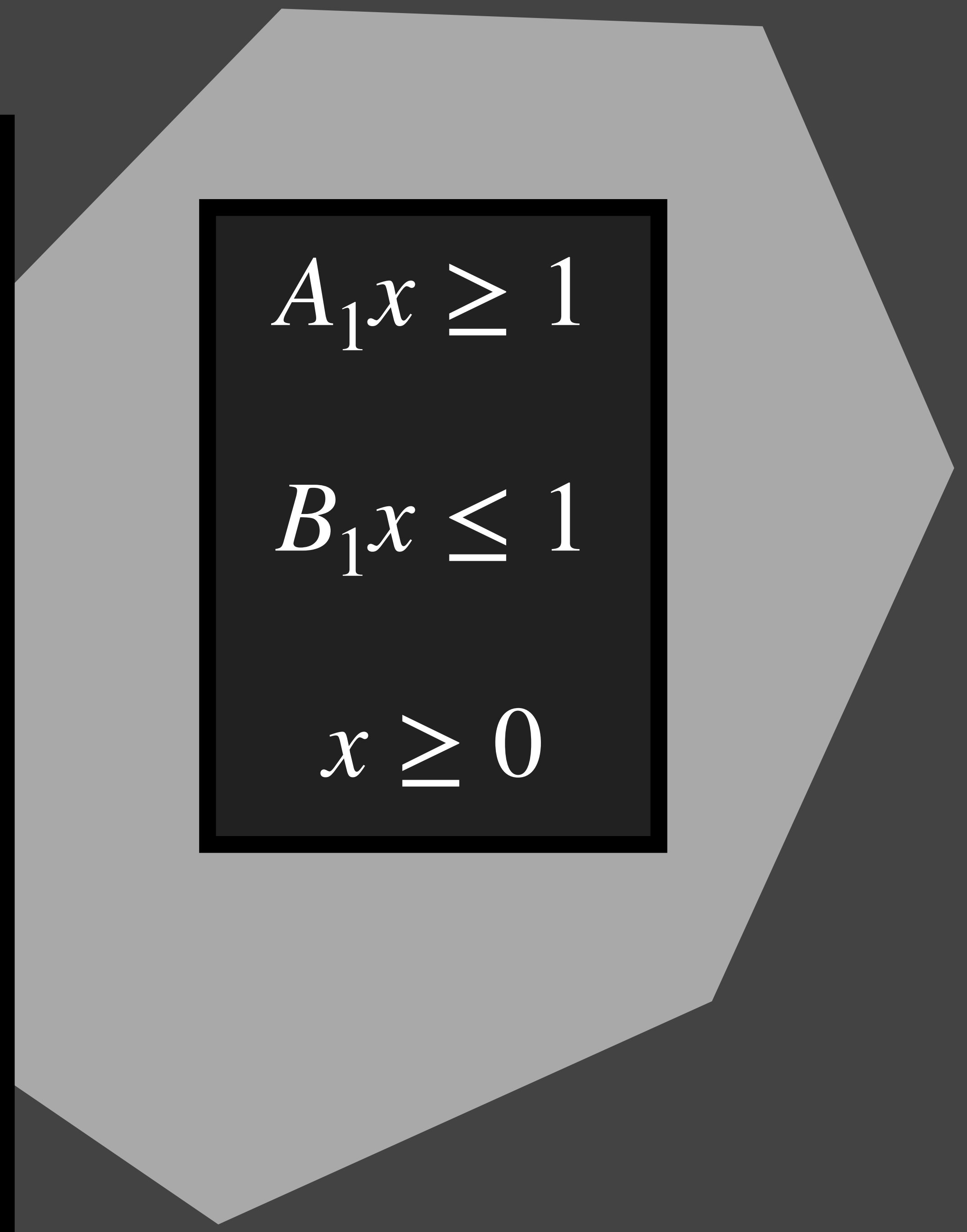


Is There a Theory to Build?

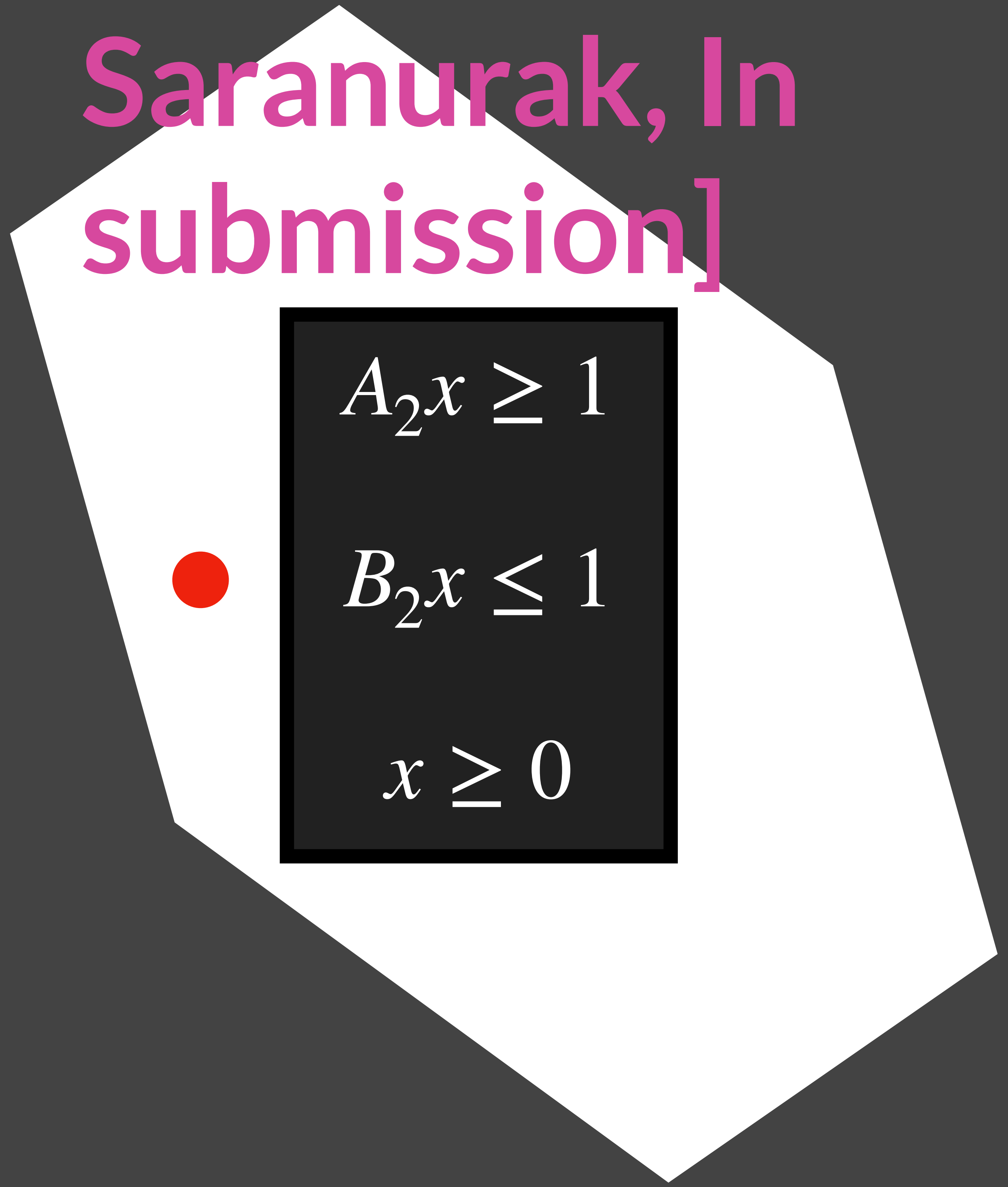
(Yes)

Theorem [Bhattacharya, Buchbinder, L., Saranurak, In submission]:

Dynamic Linear Programming with movement $O(\log n) \cdot \text{OPT}$.



[Bhattacharya, Buchbinder, L., Saranurak, In submission]



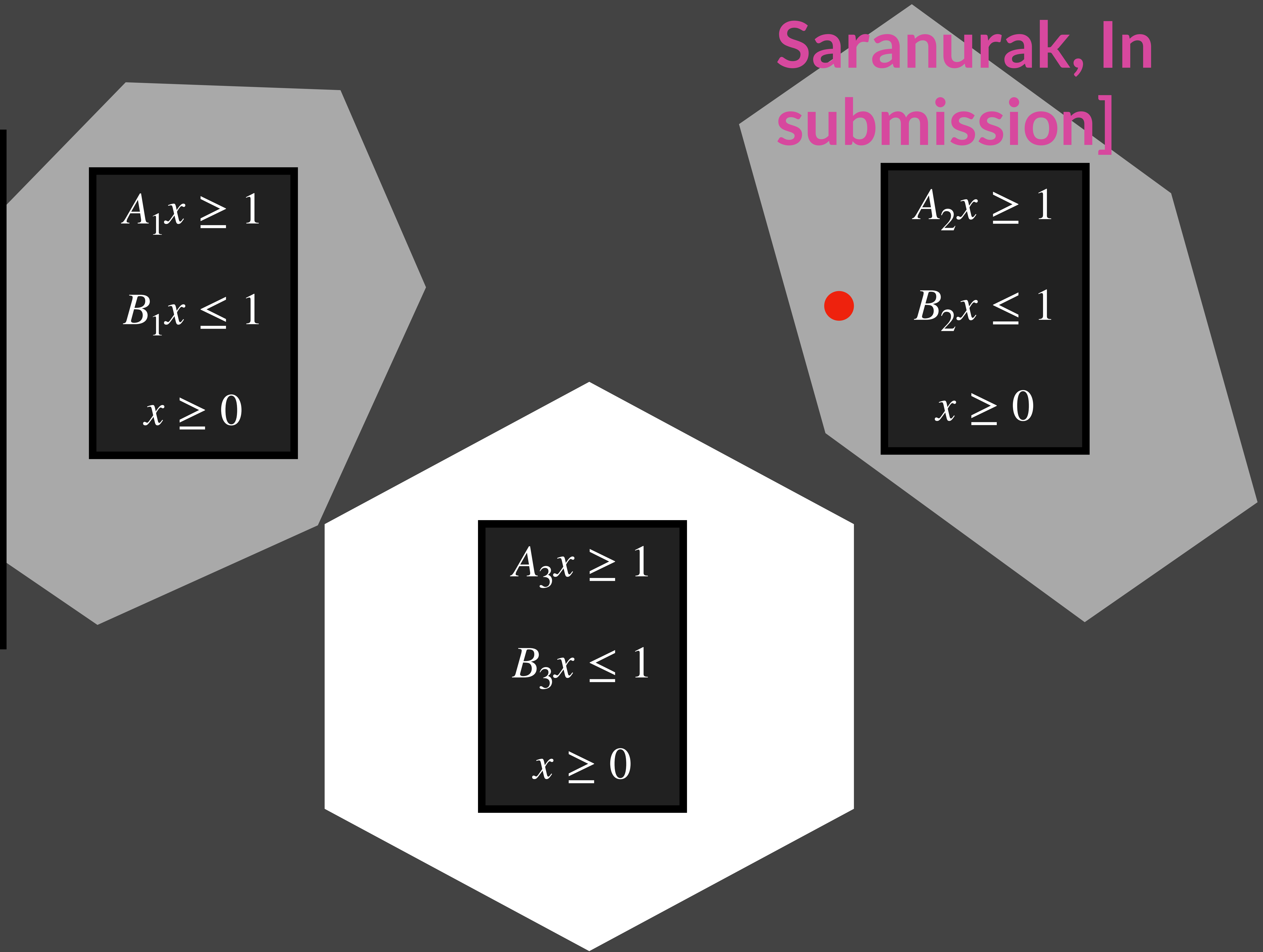
Is There a Theory to Build?

(Yes)

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

Theorem [Bhattacharya, Buchbinder, L., Saranurak, In submission]:

Dynamic Linear Programming with movement $O(\log n) \cdot \mathrm{OPT}$.



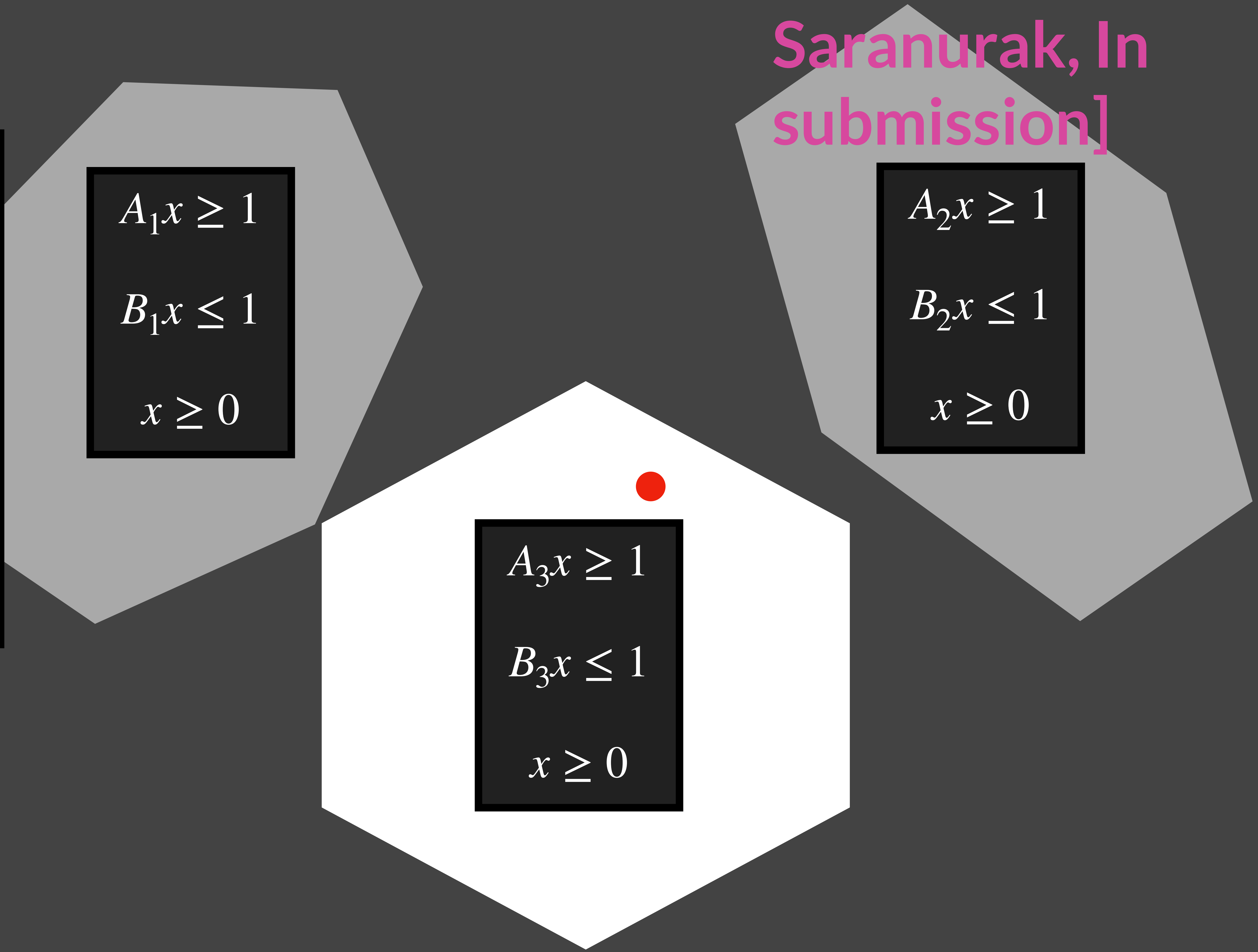
Is There a Theory to Build?

(Yes)

Theorem [Bhattacharya, Buchbinder, L., Saranurak, In submission]:

Dynamic Linear Programming with movement $O(\log n) \cdot \mathrm{OPT}$.

[Bhattacharya, Buchbinder, L., Saranurak, In submission]



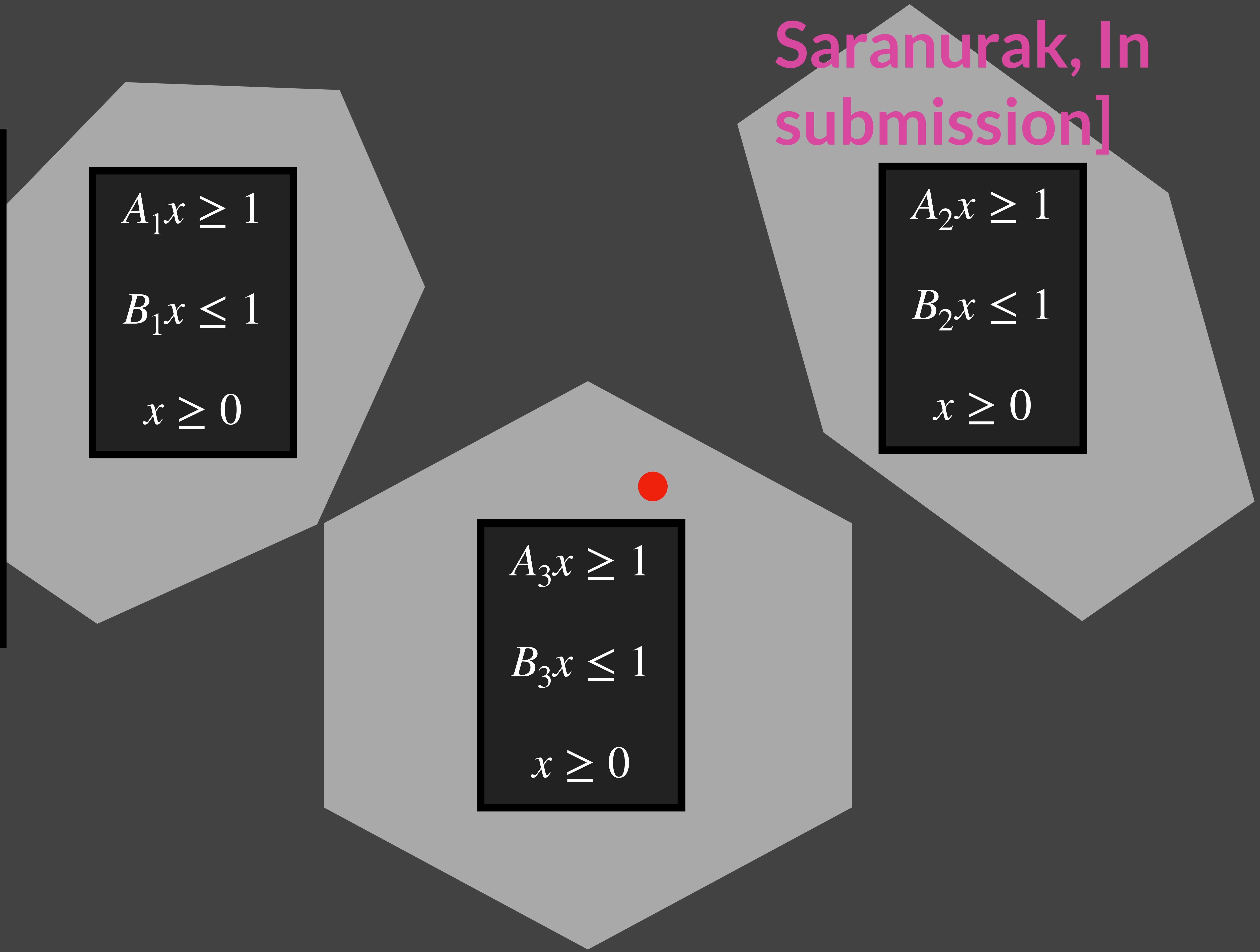
Is There a Theory to Build?

(Yes)

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

Theorem [Bhattacharya, Buchbinder, L., Saranurak, In submission]:

Dynamic Linear Programming with movement $O(\log n) \cdot \mathrm{OPT}$.



Is There a Theory to Build?

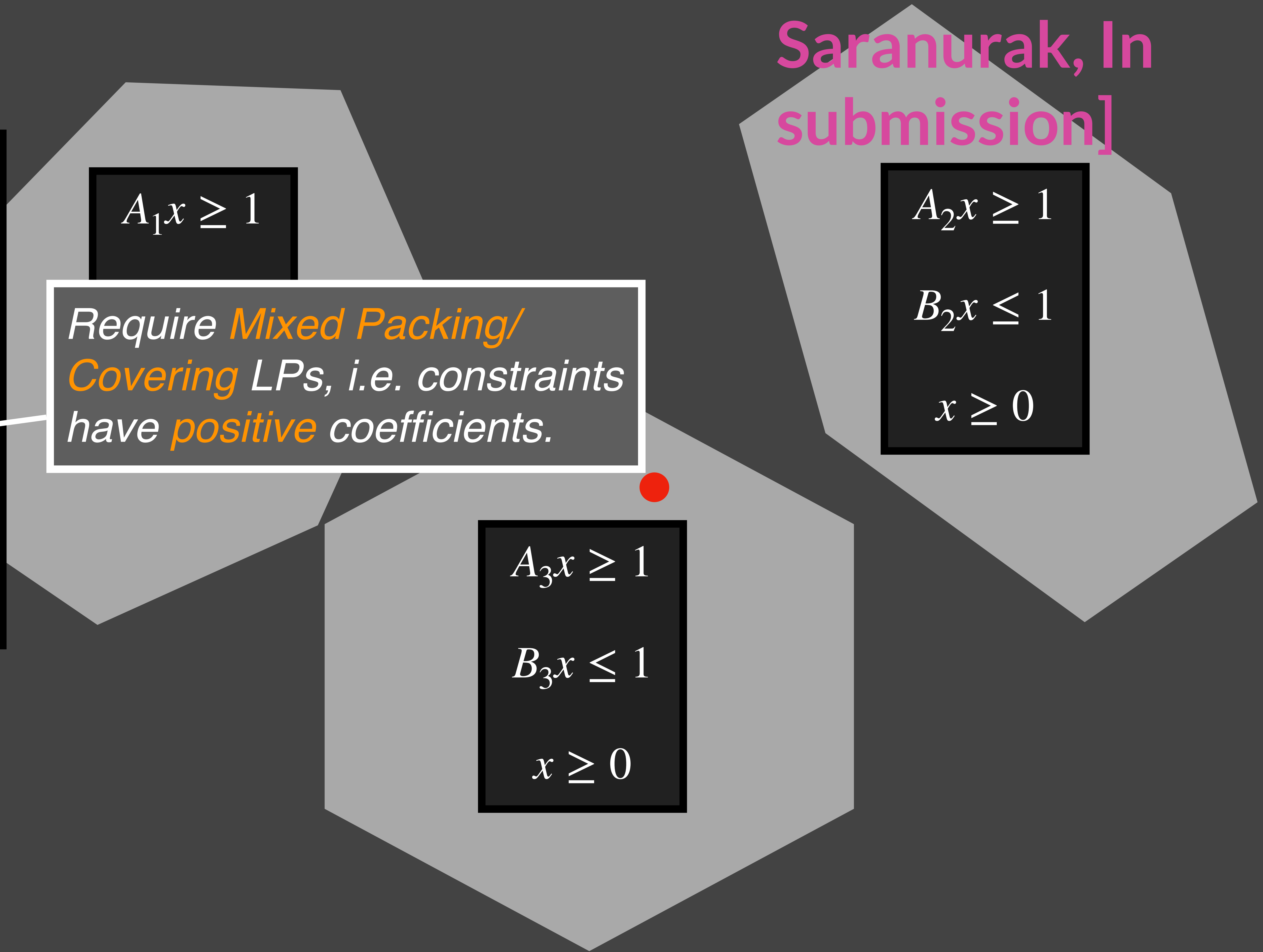
(Yes)

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

Theorem [Bhattacharya, Buchbinder, L., Saranurak, In submission]:

Dynamic Linear Programming with movement $O(\log n) \cdot$ OPT.

Require Mixed Packing/ Covering LPs, i.e. constraints have positive coefficients.



Is There a Theory to Build?

(Yes)

Theorem [Bhattacharya, Buchbinder, L., Saranurak, In submission]:

Dynamic Linear Programming with movement $O(\log n) \cdot$ OPT.

Require Mixed Packing/ Covering LPs, i.e. constraints have positive coefficients.

$A_1 x \geq 1$

$A_2 x \geq 1$

$B_2 x \leq 1$

$x \geq 0$

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

$A_3 x \geq 1$

$B_3 x \leq 1$

$x \geq 0$

Rounding gives improved results for Dynamic Set Cover, Load Balancing, Matching, Minimum Spanning Tree.

Is There a Theory to Build?

(Yes)

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

Theorem [Bhattacharya, Buchbinder, L., Saranurak, In submission]:

Dynamic Linear Programming with movement $O(\log n) \cdot \text{OPT}$.

Require Mixed Packing/ Covering LPs, i.e. constraints have positive coefficients.

$A_1 x \geq 1$

$A_2 x \geq 1$

$B_2 x \leq 1$

$x \geq 0$

$A_3 x \geq 1$

$B_3 x \leq 1$

$x \geq 0$

Rounding gives improved results for Dynamic Set Cover, Load Balancing, Matching, Minimum Spanning Tree.

Technical Ingredient: *Max Entropy Principle.*

Is There a Theory to Build?

(Yes)

Theorem [Bhattacharya, Buchbinder, L., Saranurak, In submission]:

Dynamic Linear Programming with movement $O(\log n) \cdot$ OPT.

Require *Mixed Packing/ Covering LPs, i.e. constraints have positive coefficients.*

$A_1 x \geq 1$

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

$A_2 x \geq 1$

$B_2 x \leq 1$

$x \geq 0$

$A_3 x \geq 1$

$B_3 x \leq 1$

$x \geq 0$

Optimal!

Rounding gives improved results for Dynamic Set Cover, Load Balancing, Matching, Minimum Spanning Tree.

Technical Ingredient: *Max Entropy Principle.*

Take Away II

[Gupta L. FOCS 20]

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

Q: Can we understand recourse/approximation tradeoffs?

Take Away II

[Gupta L. FOCS 20]

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

Q: Can we understand recourse/approximation tradeoffs?

A1: Get optimal tradeoff for Submodular Cover class.

Take Away II

[Gupta L. FOCS 20]

[Bhattacharya, Buchbinder, L., Saranurak, In submission]

Q: Can we understand recourse/approximation tradeoffs?

A1: Get optimal tradeoff for Submodular Cover class.

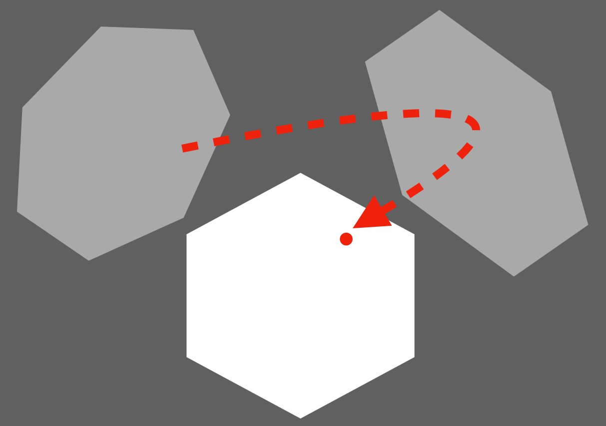
A2: Get stable Dynamic analogs of fundamental algorithmic primitive, Linear Programming.

Outline

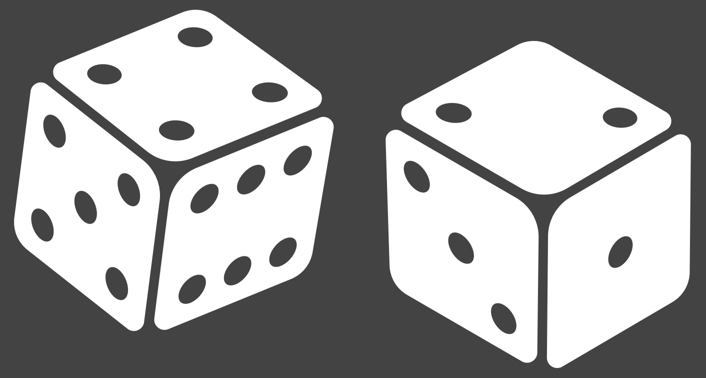
Theme I — Submodular Optimization

f(🍕 | 🥕) ≥ *f*(🍕 | 🥕 , 🍩)

Theme II — Stable Algorithms



Theme III — Beyond Worst-Case Analysis



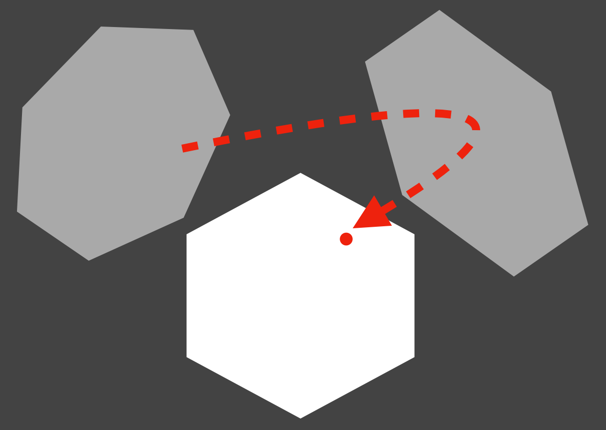
Conclusion

Outline

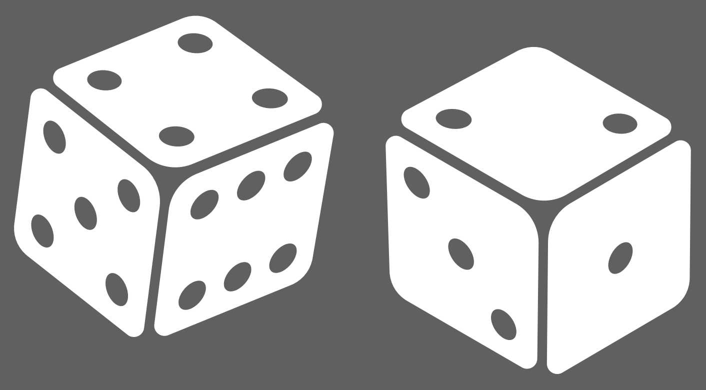
Theme I — Submodular Optimization

$f(🍕 \mid 🥕) \geq f(🍕 \mid 🥕, 🍩)$

Theme II — Stable Algorithms



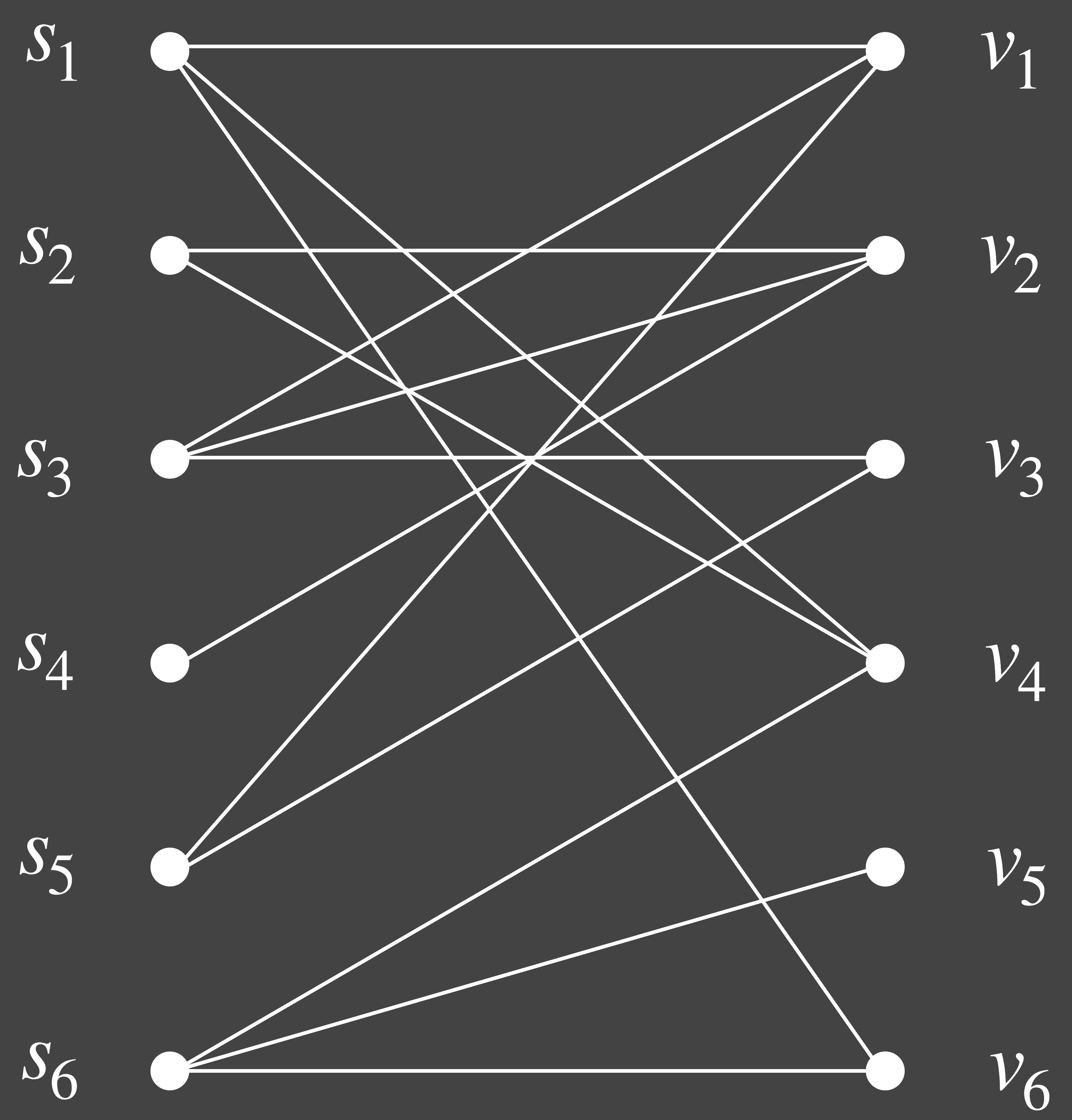
Theme III — Beyond Worst-Case Analysis



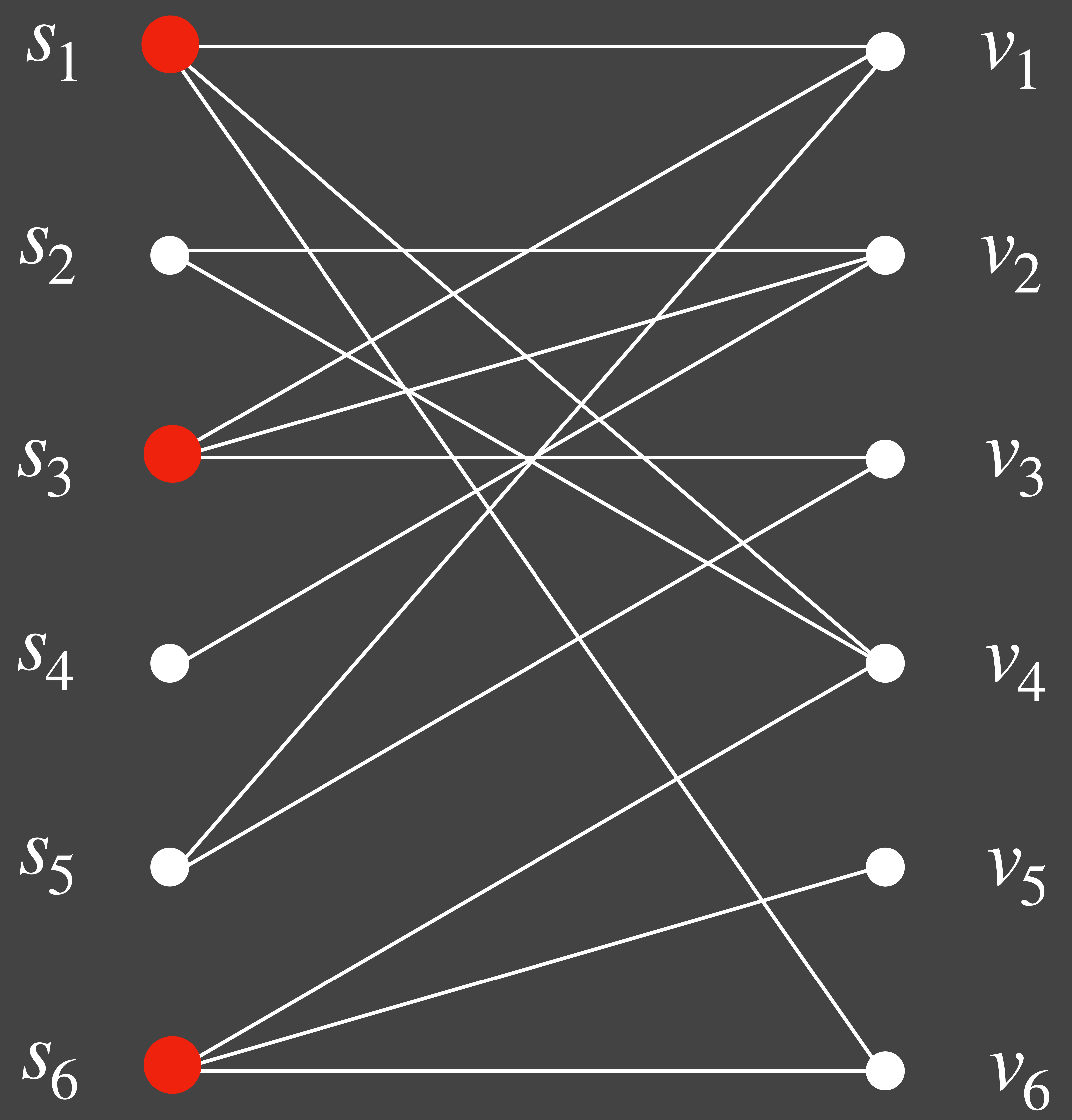
Conclusion

Theme III — Beyond Worst-Case Analysis

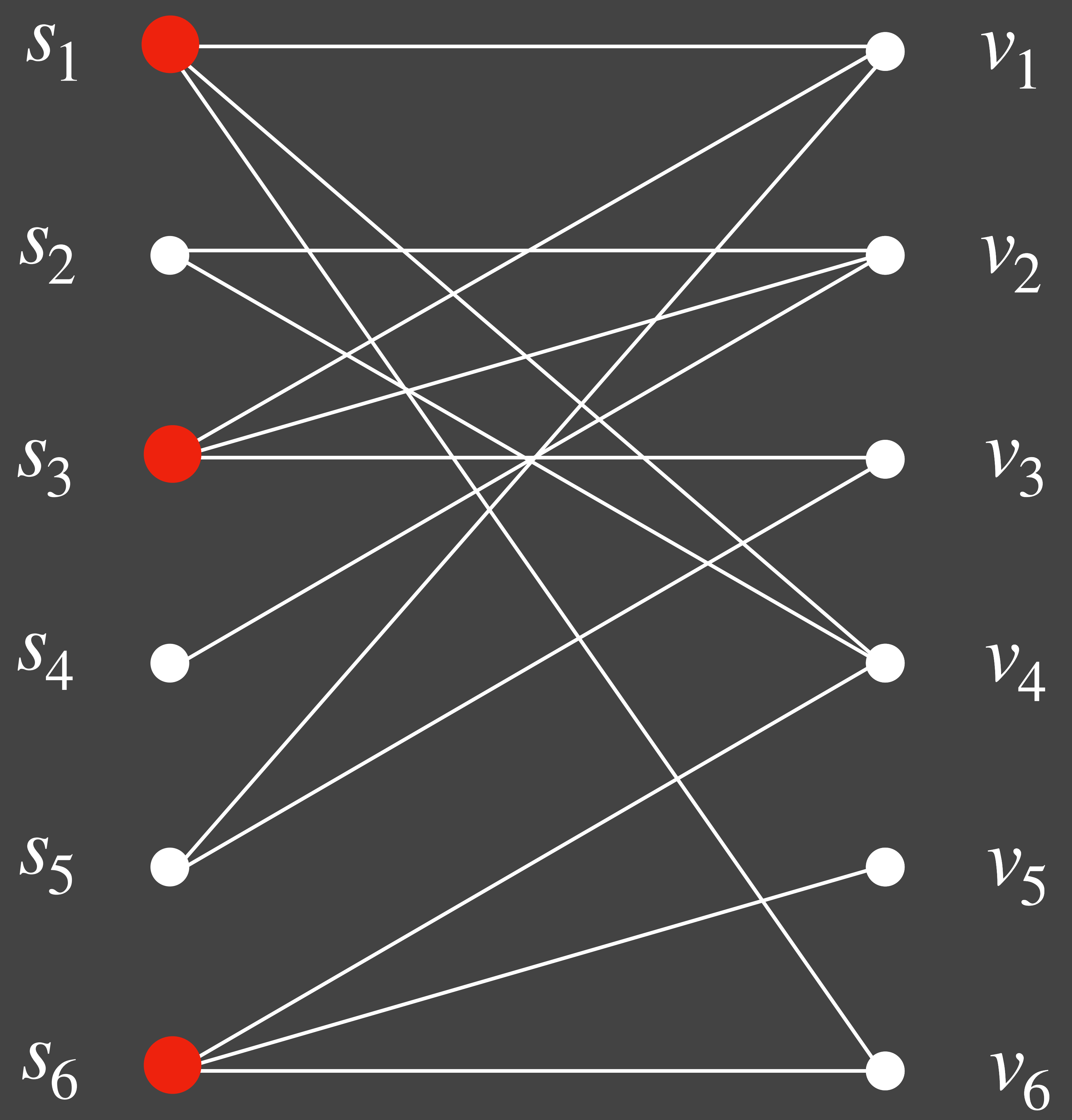
Set Cover



Set Cover

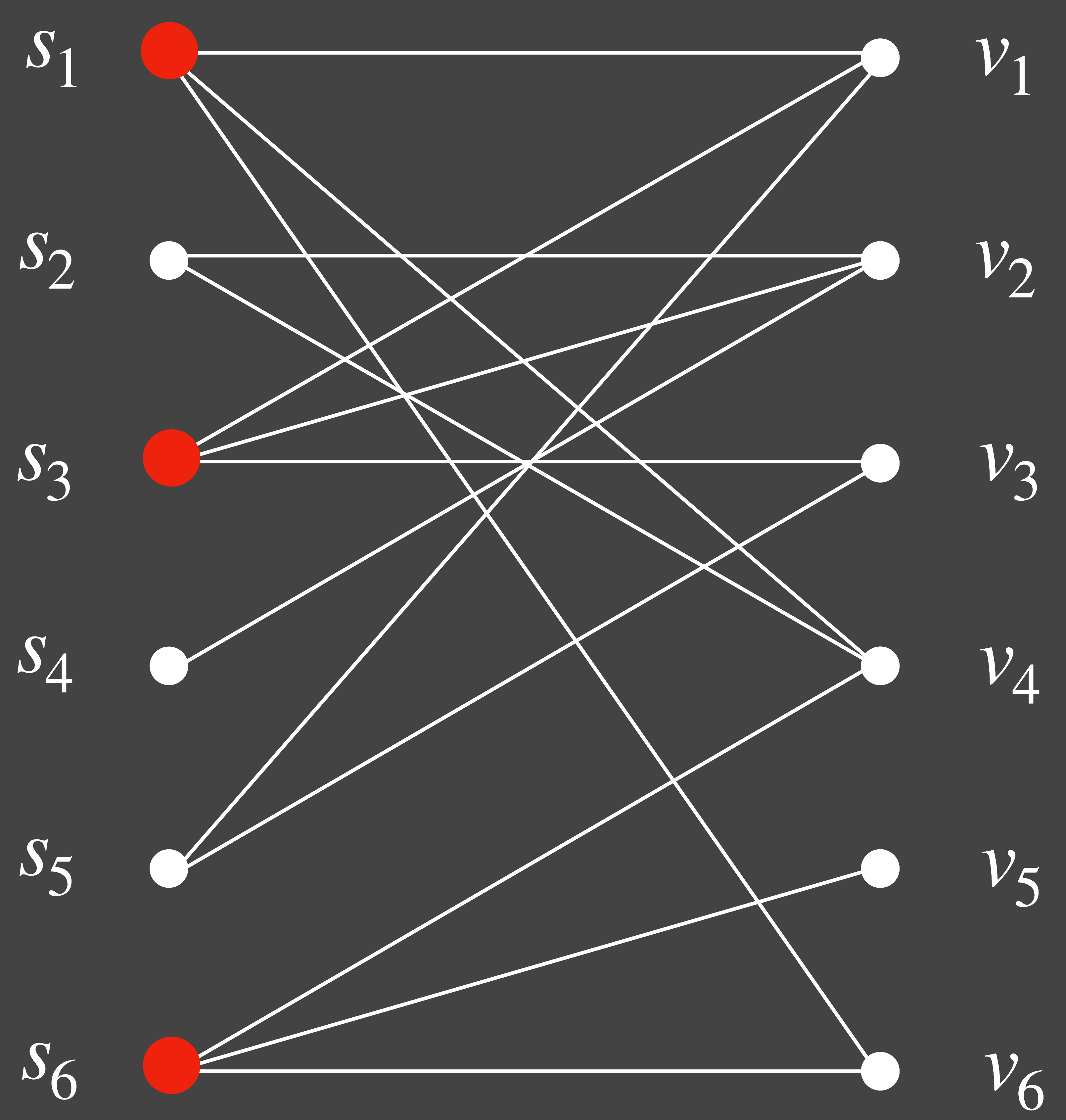


Set Cover



Approximation:
$O(\log n)$
[Johnson 74],
[Lovasz 75],
[Chvatal 79]

Set Cover



Approximation:
$O(\log n)$
[Johnson 74],
[Lovasz 75],
[Chvatal 79]

Optimal!
(in poly time)

Online Set Cover [Alon Awerbuch Azar Buchbinder Naor 03]

s_1

s_2

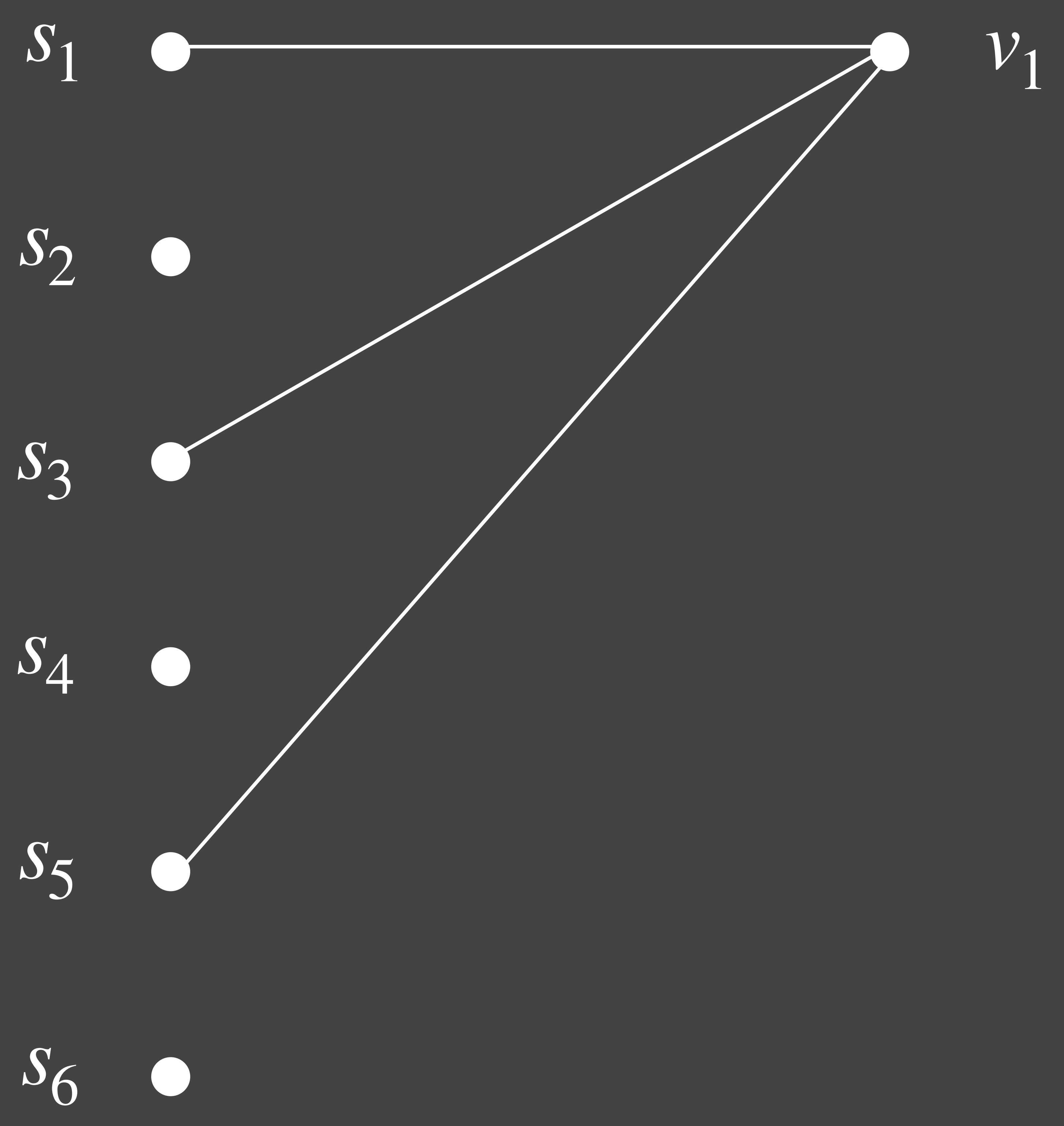
s_3

s_4

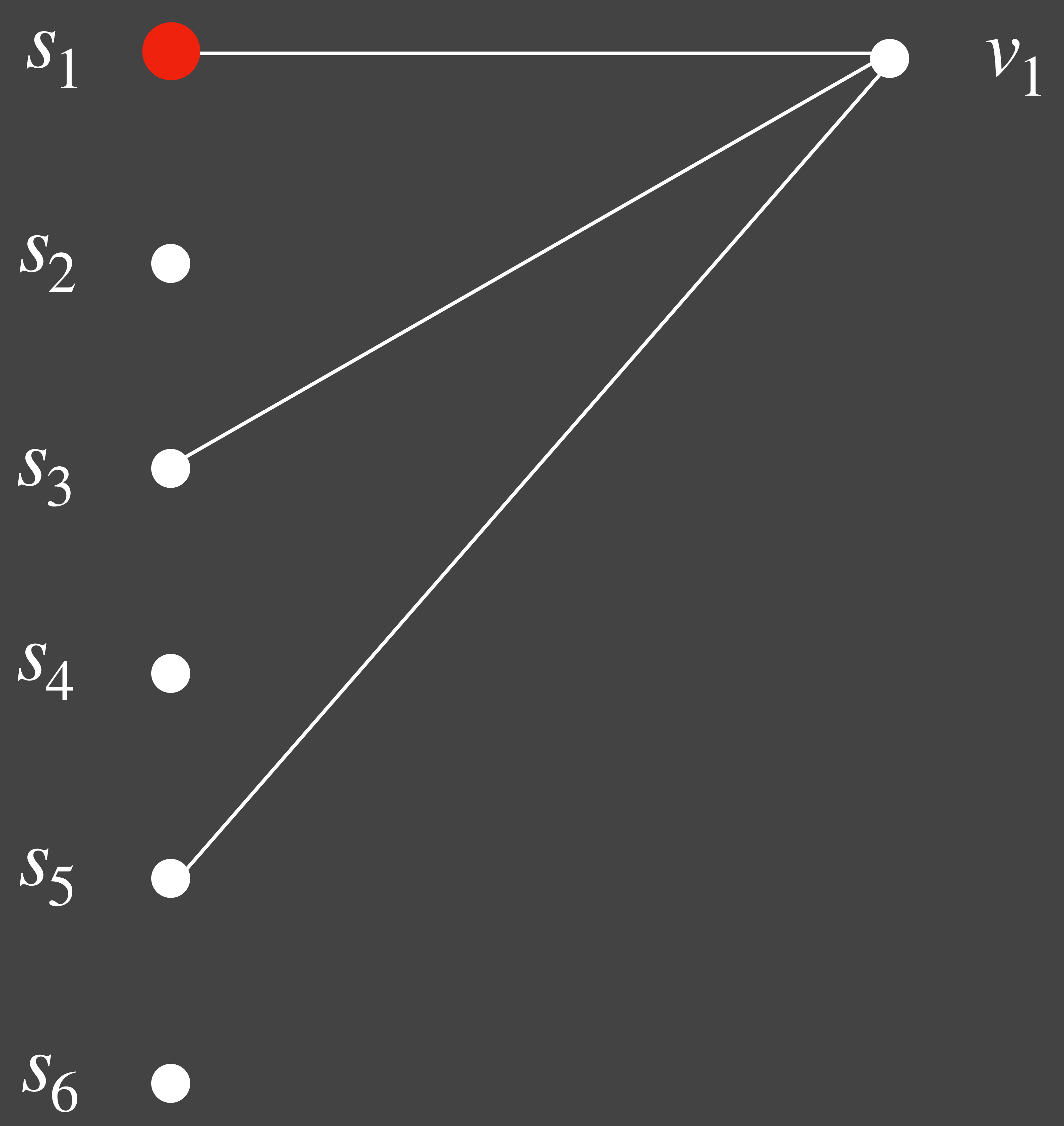
s_5

s_6

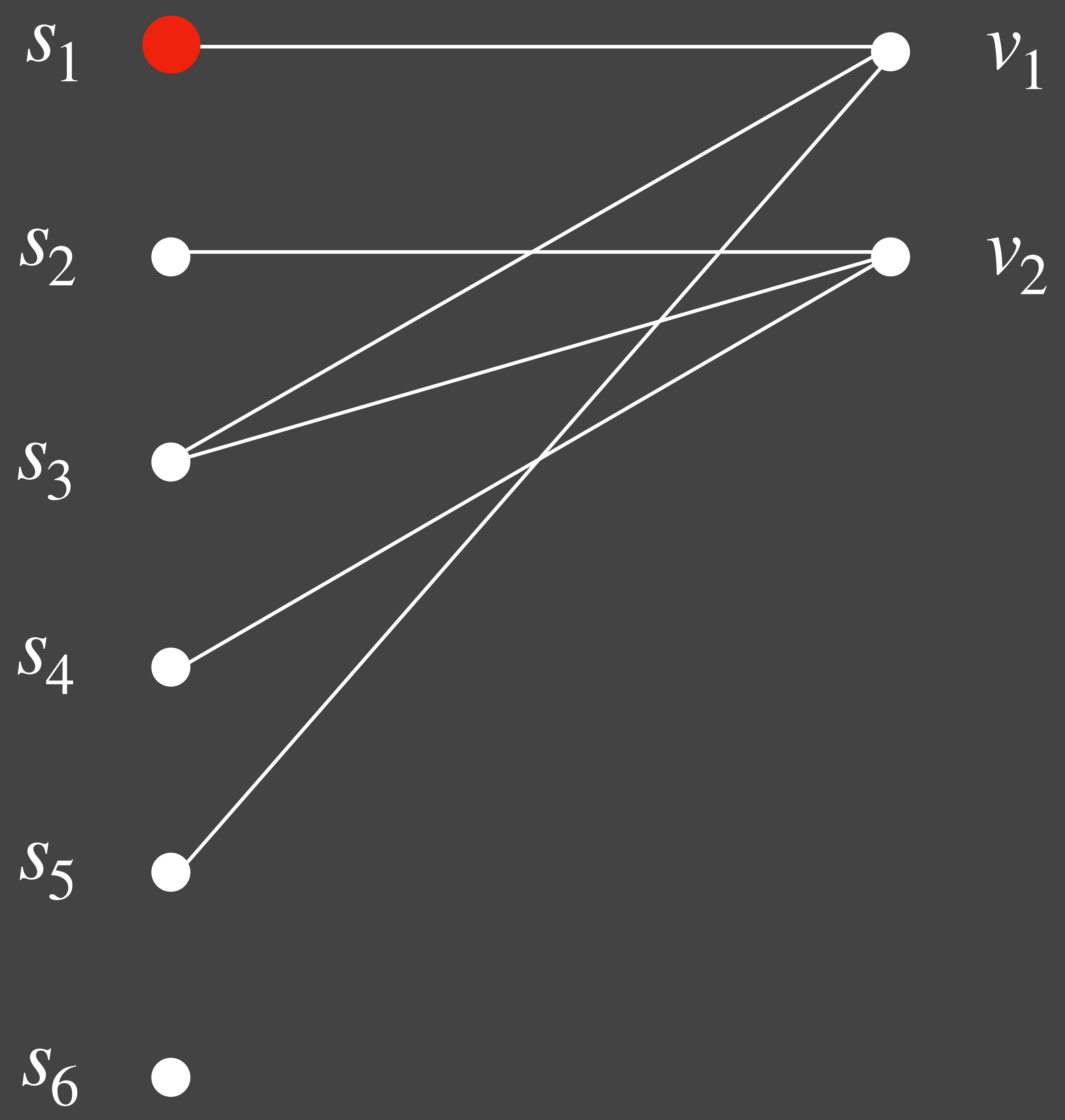
Online Set Cover [Alon Awerbuch Azar Buchbinder Naor 03]



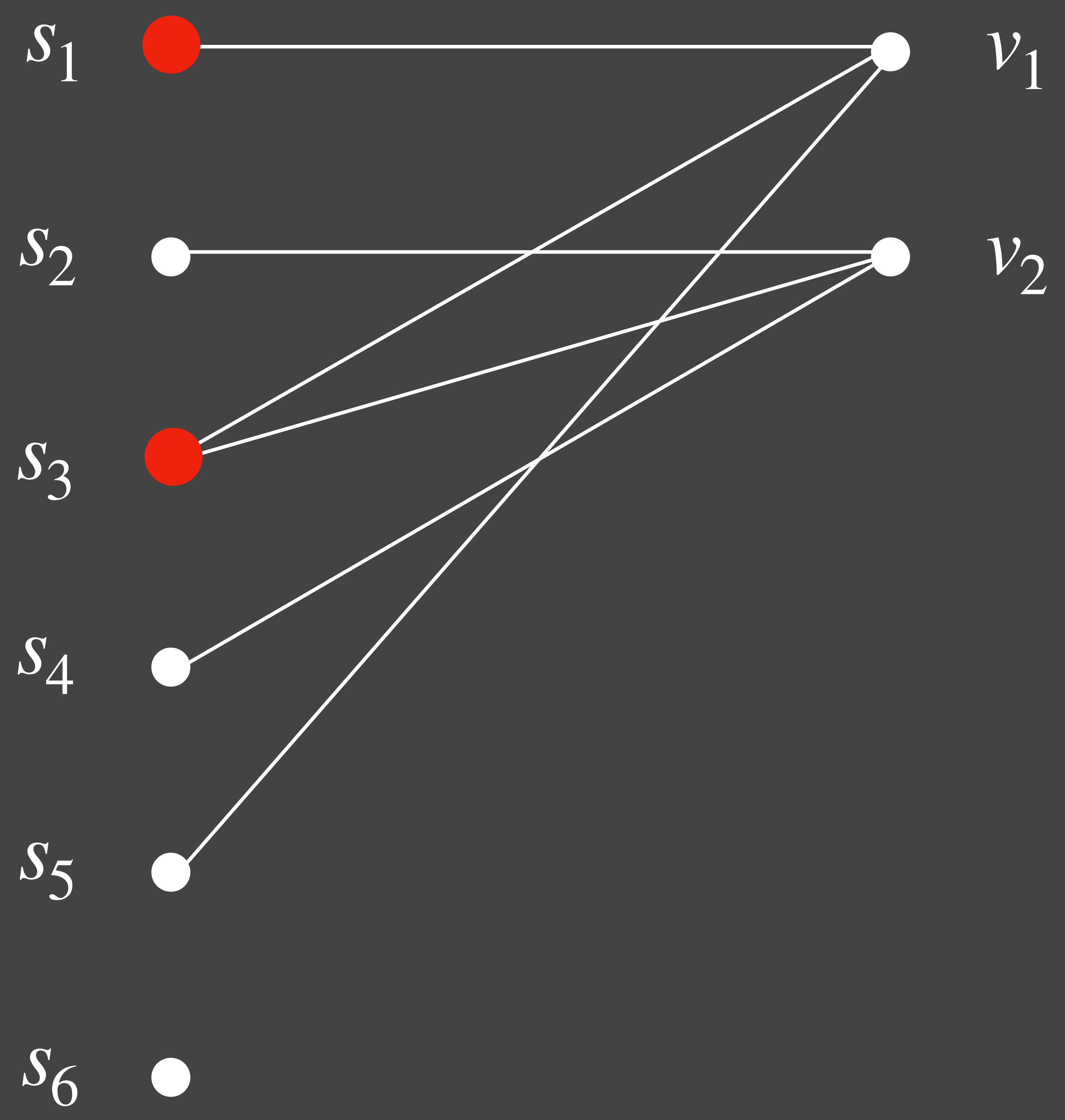
Online Set Cover [Alon Awerbuch Azar Buchbinder Naor 03]



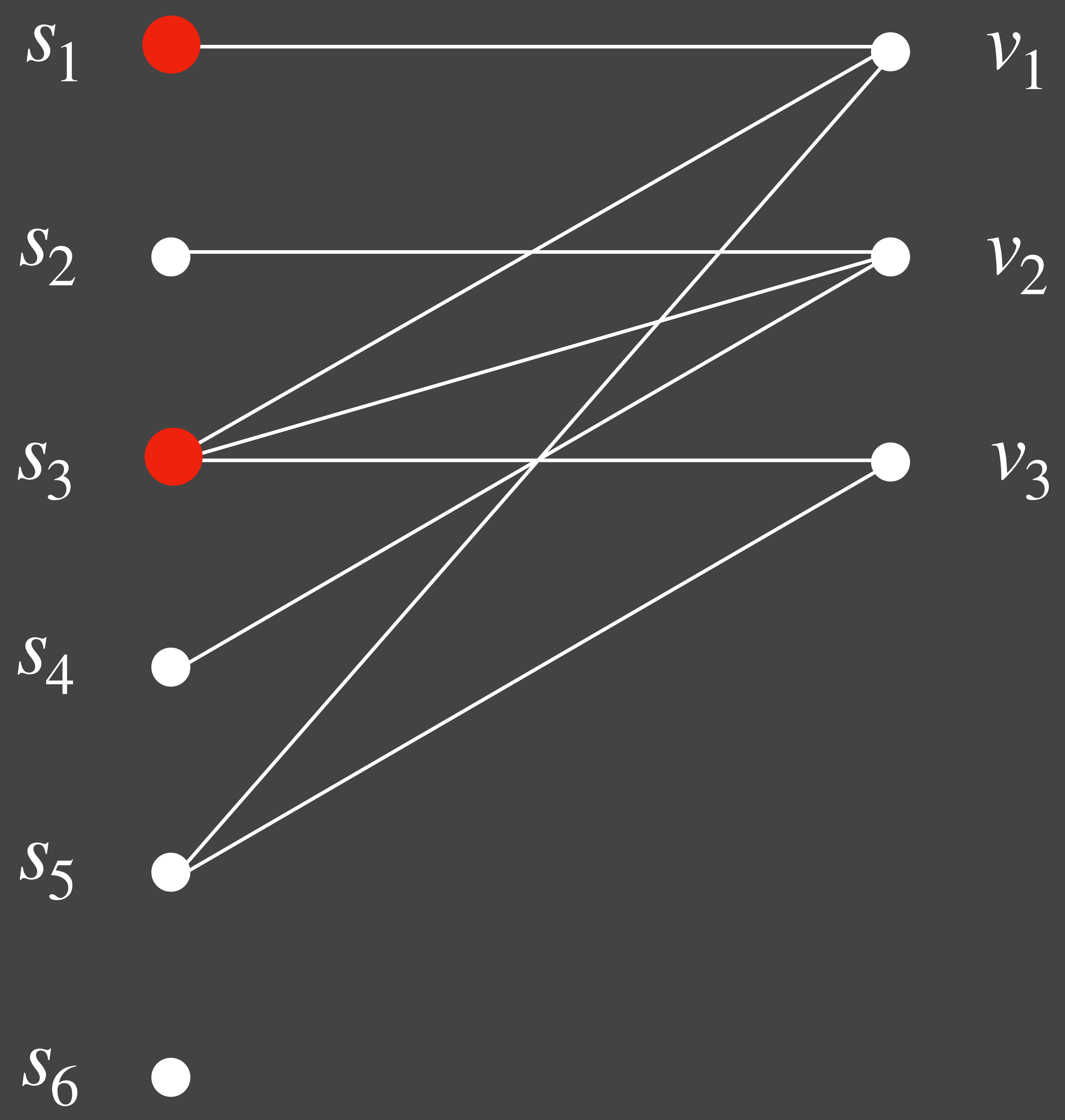
Online Set Cover [Alon Awerbuch Azar Buchbinder Naor 03]



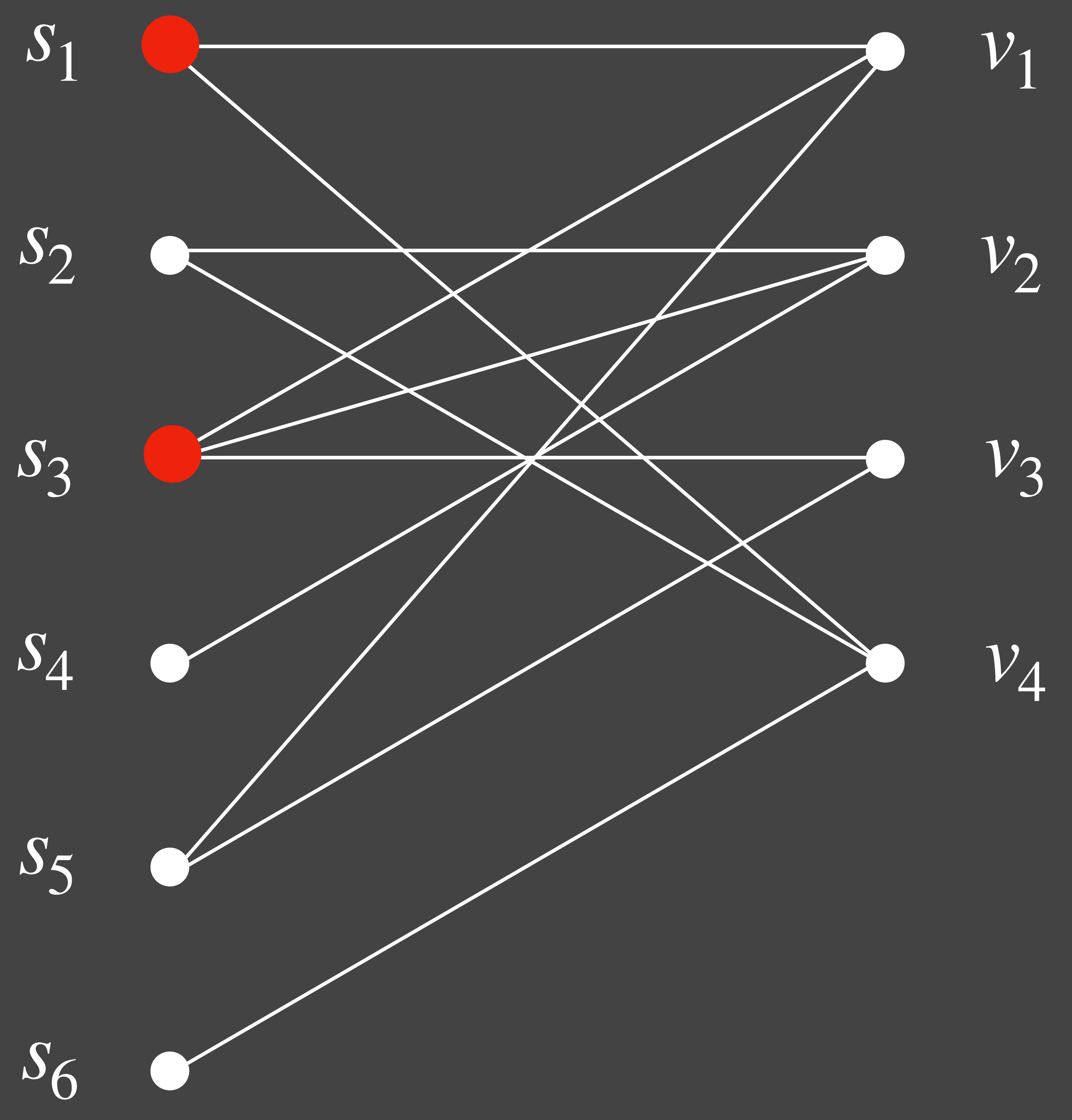
Online Set Cover [Alon Awerbuch Azar Buchbinder Naor 03]



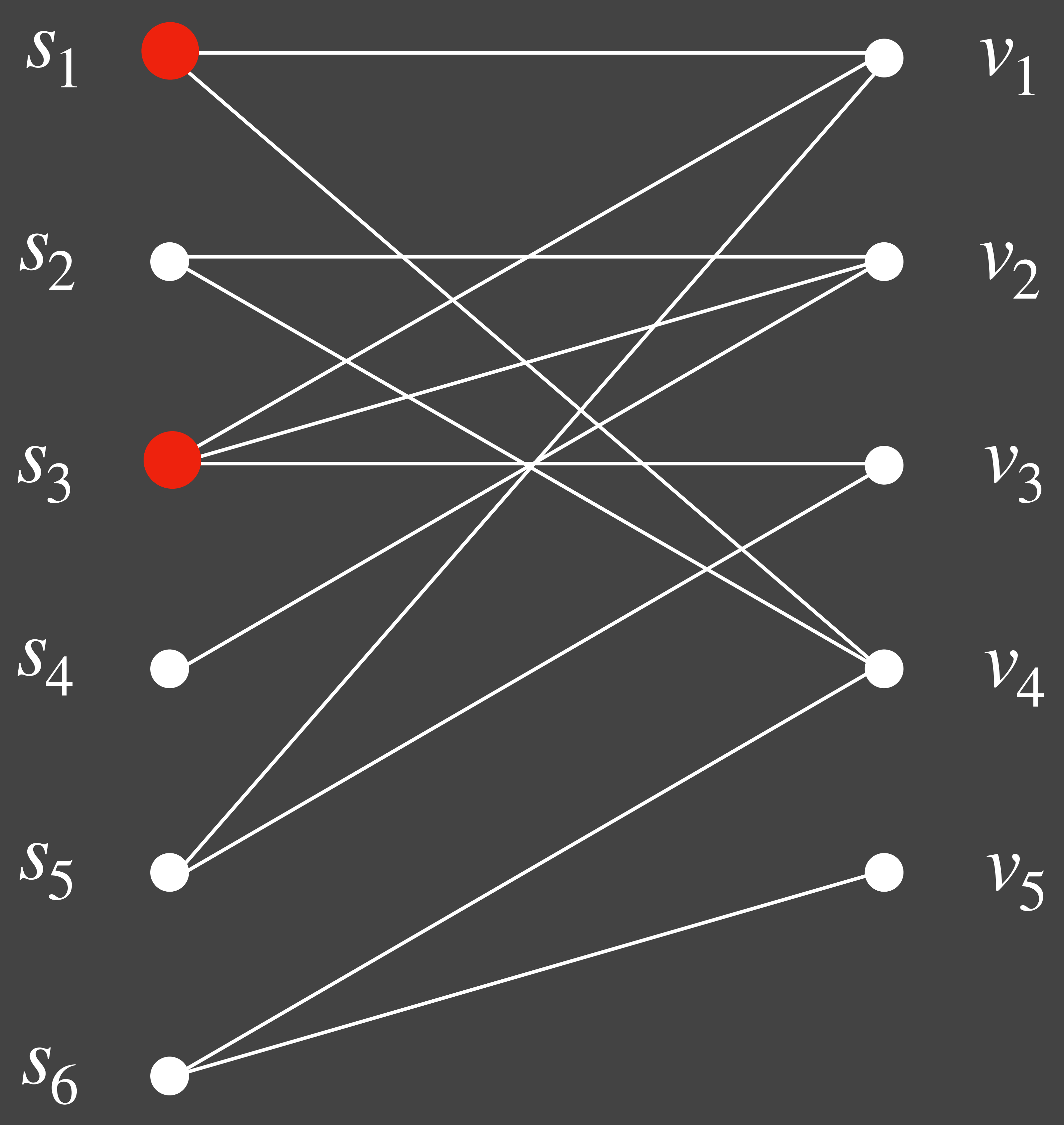
Online Set Cover [Alon Awerbuch Azar Buchbinder Naor 03]



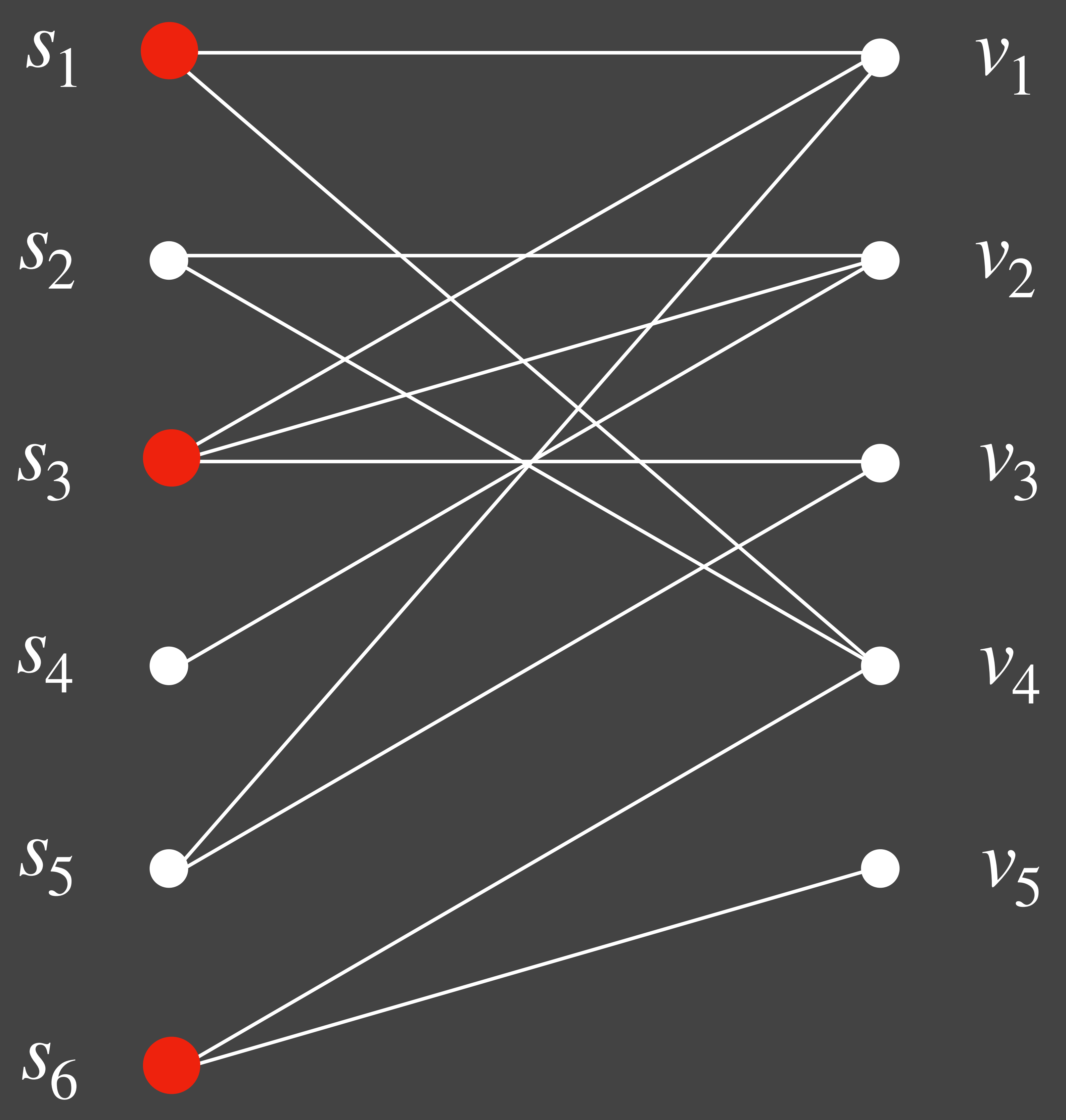
Online Set Cover [Alon Awerbuch Azar Buchbinder Naor 03]



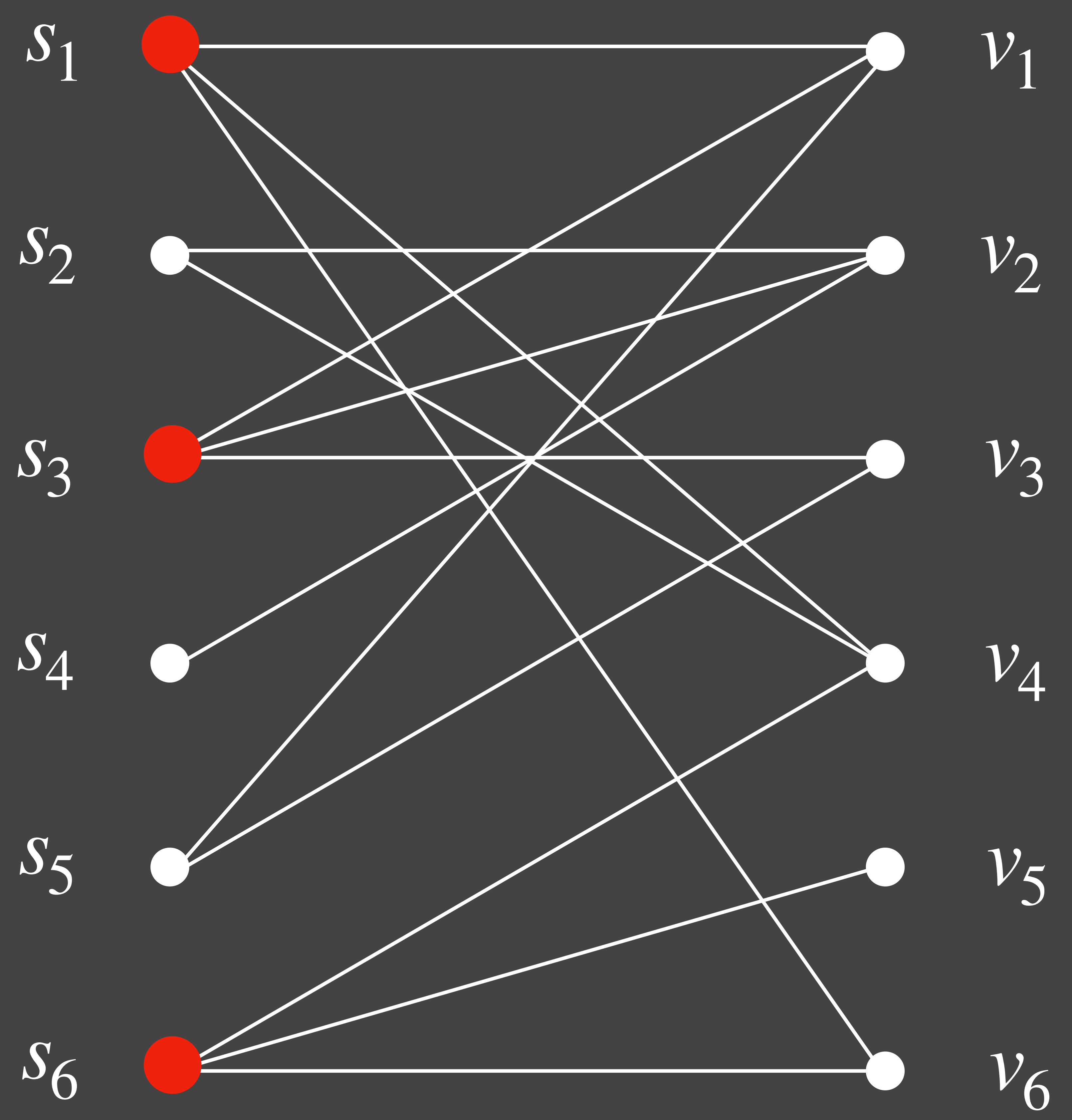
Online Set Cover [Alon Awerbuch Azar Buchbinder Naor 03]



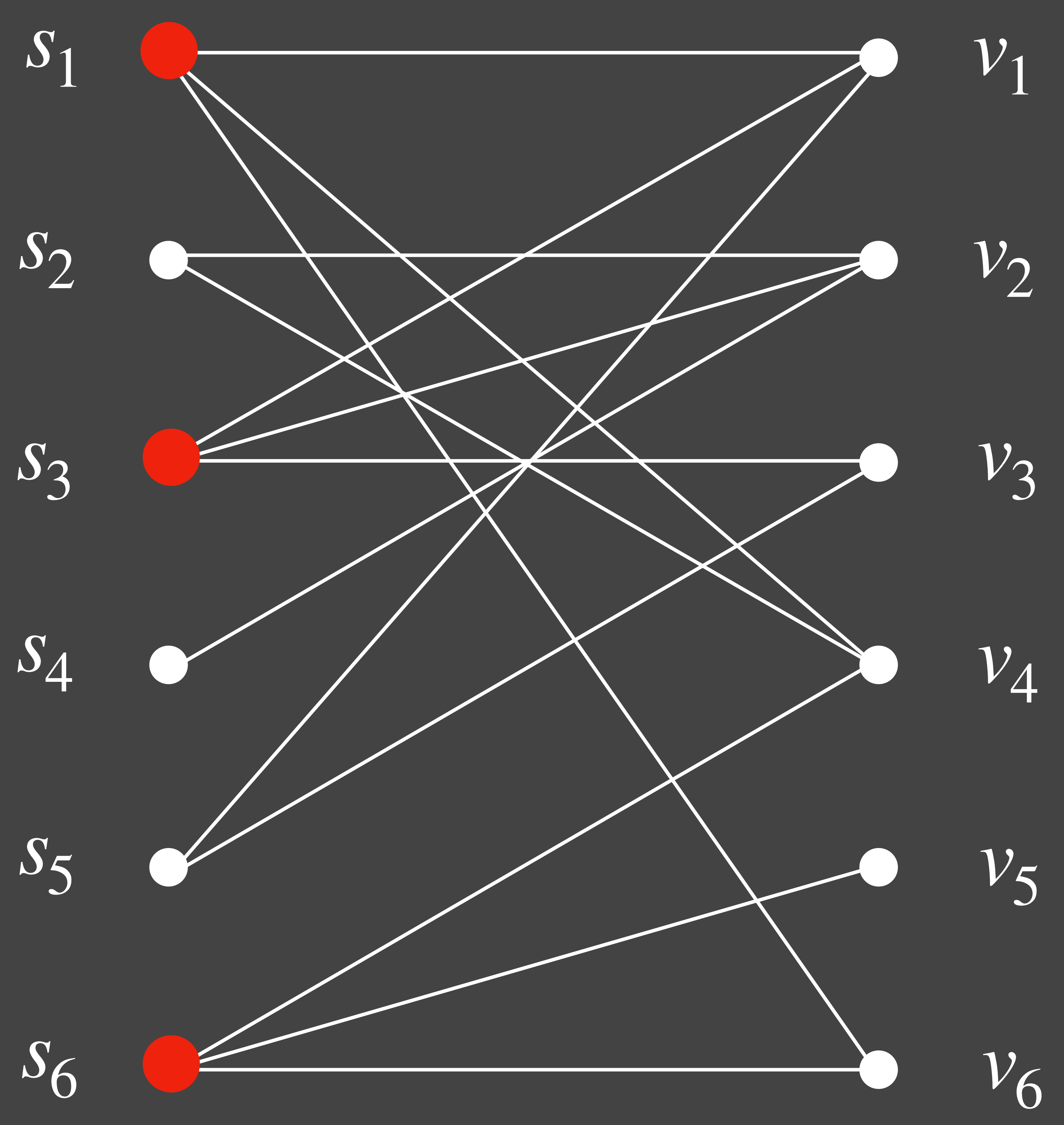
Online Set Cover [Alon Awerbuch Azar Buchbinder Naor 03]



Online Set Cover [Alon Awerbuch Azar Buchbinder Naor 03]

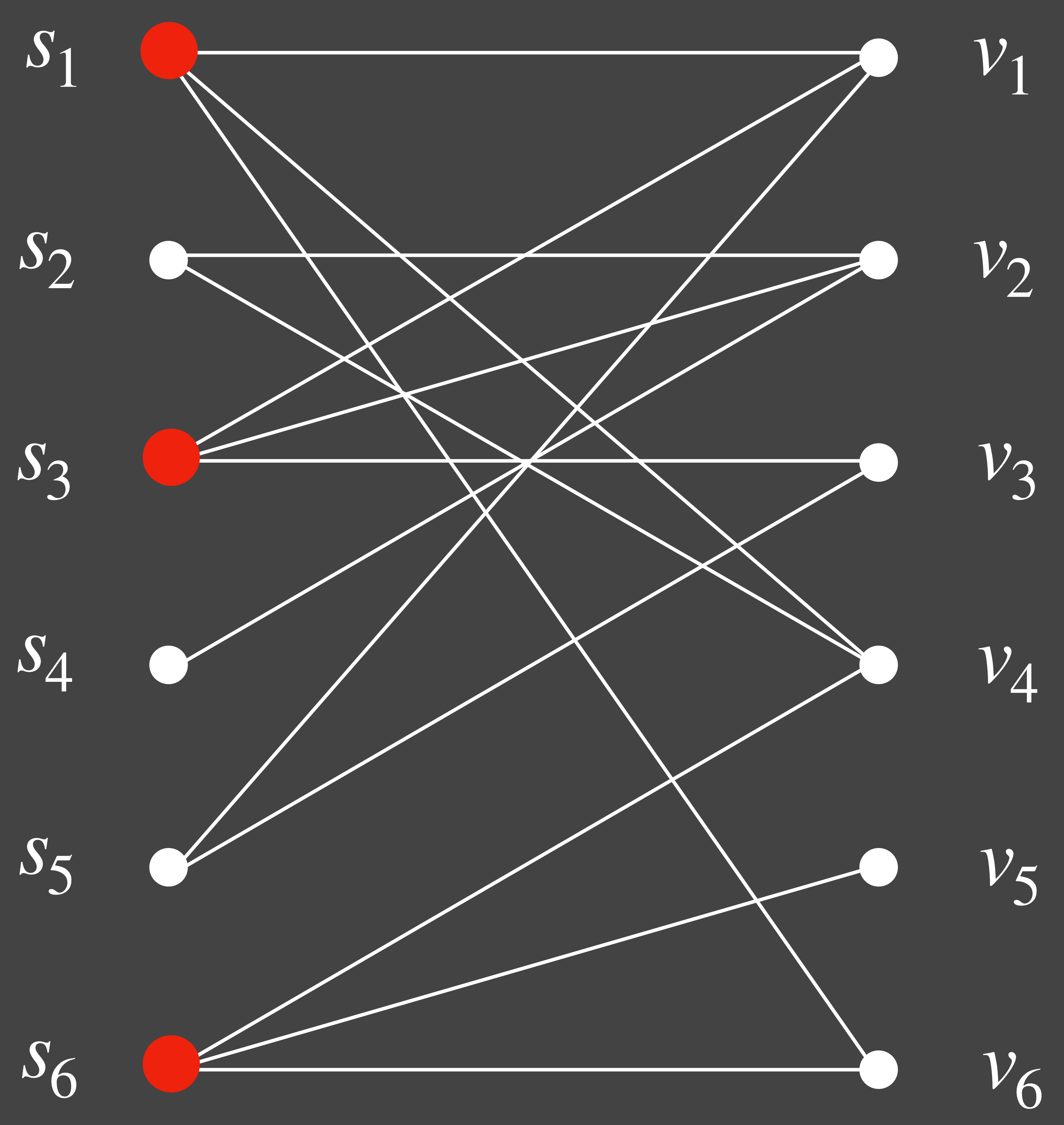


Online Set Cover [Alon Awerbuch Azar Buchbinder Naor 03]



Approximation:
$O(\log^2 n)$
[Alon+ 03]
[Buchbinder Naor 09]

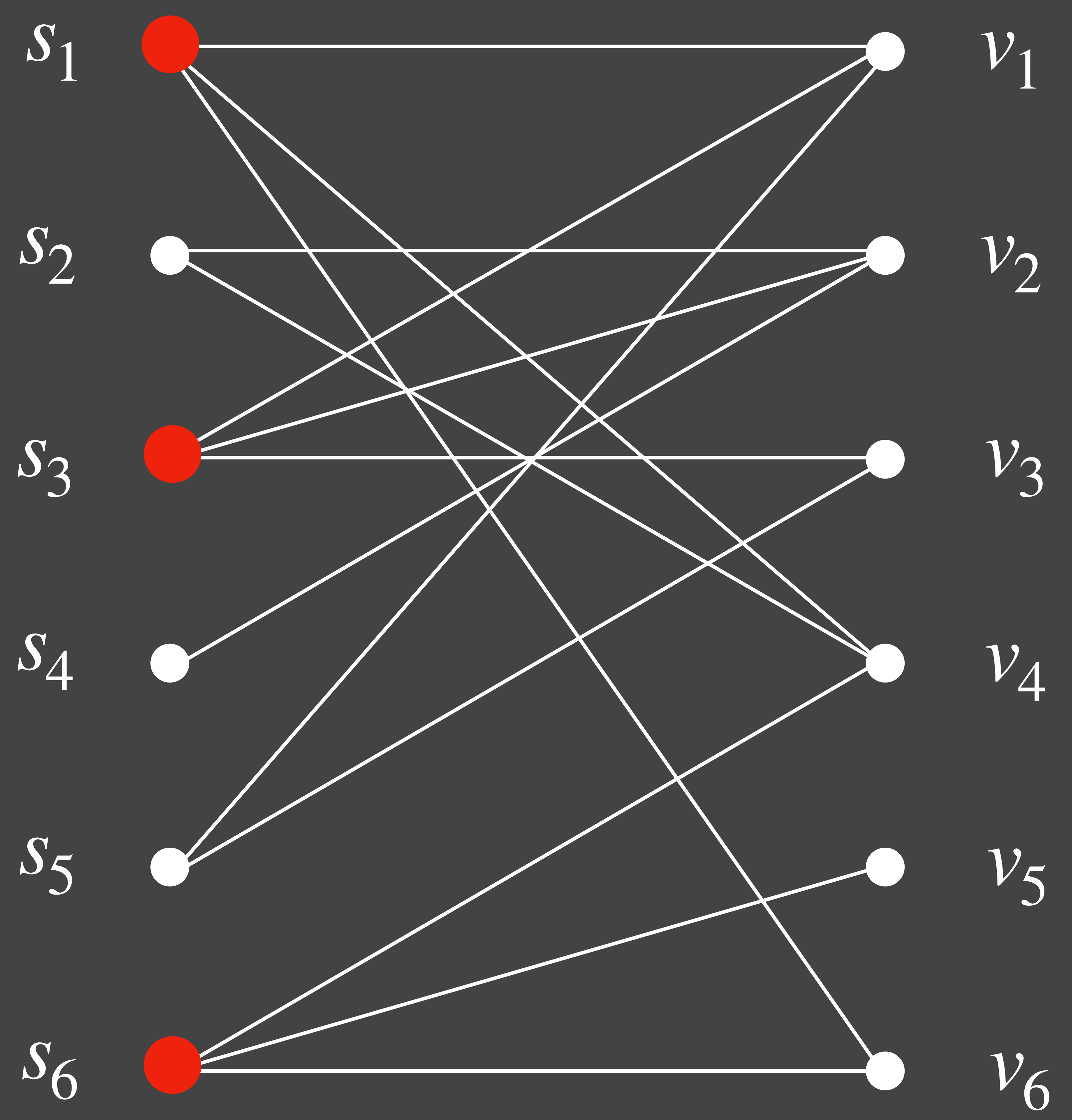
Online Set Cover [Alon Awerbuch Azar Buchbinder Naor 03]



Approximation:
$O(\log^2 n)$
[Alon+ 03]
[Buchbinder Naor 09]

Optimal!
(in poly time)

Online Set Cover [Alon Awerbuch Azar Buchbinder Naor 03]

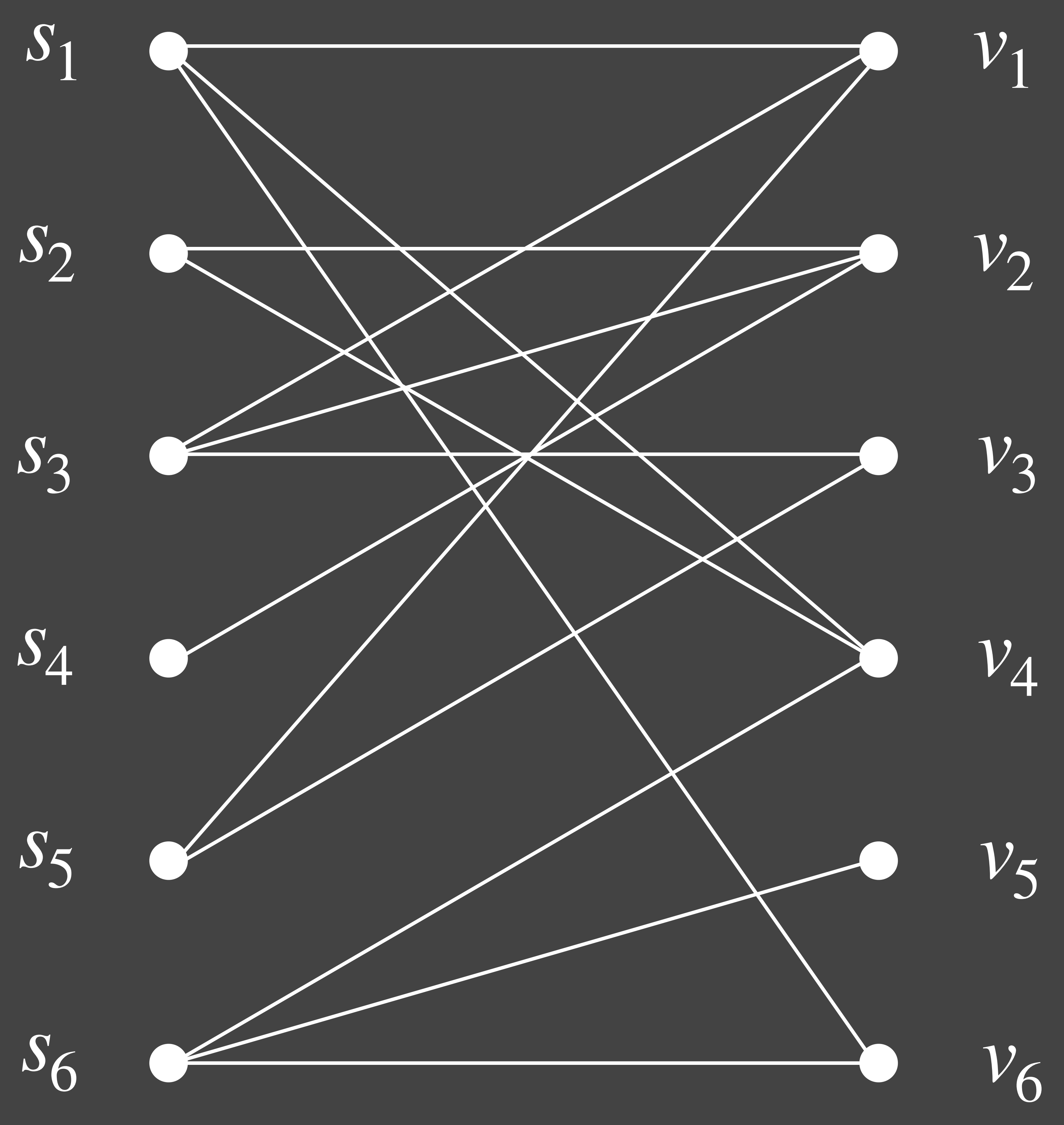


Approximation:
$O(\log^2 n)$
[Alon+ 03]
[Buchbinder Naor 09]

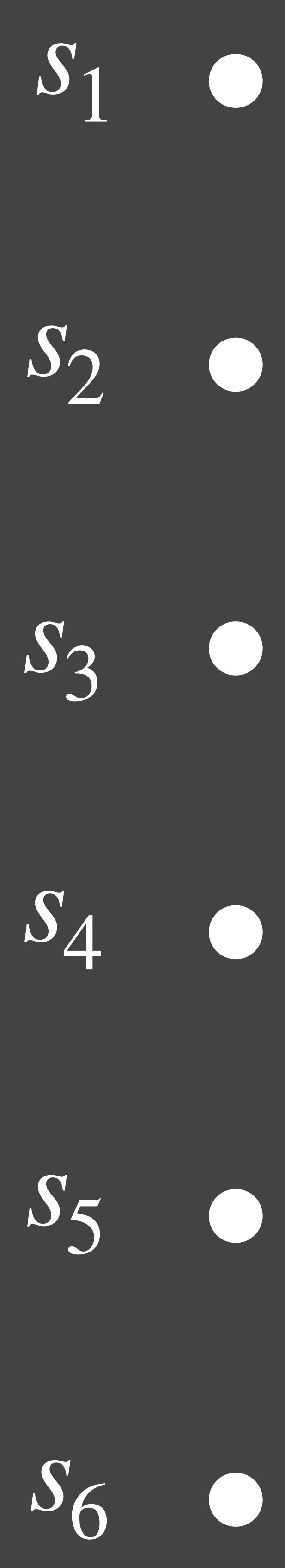
Optimal!
(in poly time)

Q: What happens beyond the worst case?

Relaxation 1: Random Order (RO)



Relaxation 1: Random Order (RO)



Relaxation 1: Random Order (RO)

s_1 ●

s_2 ●

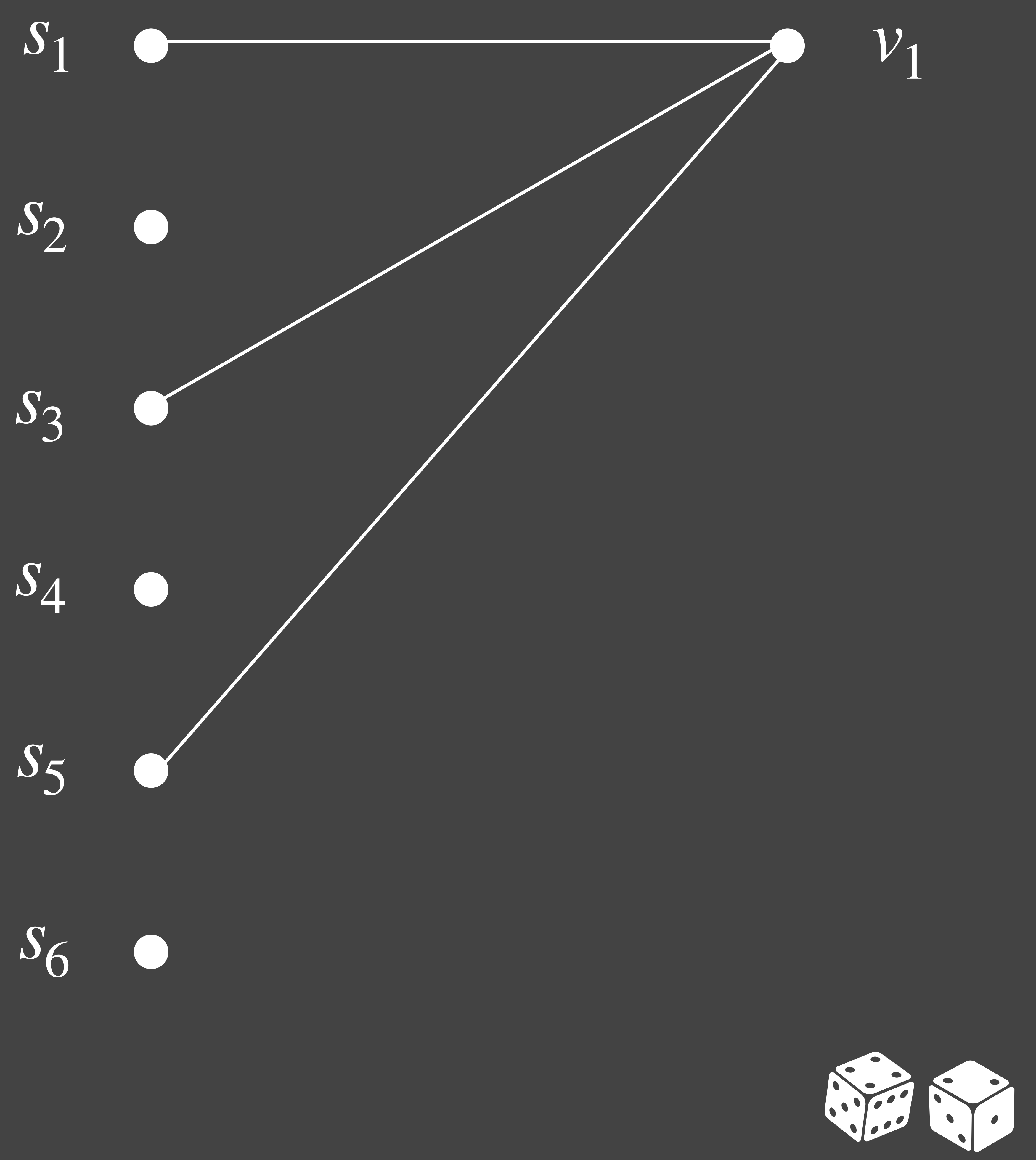
s_3 ●

s_4 ●

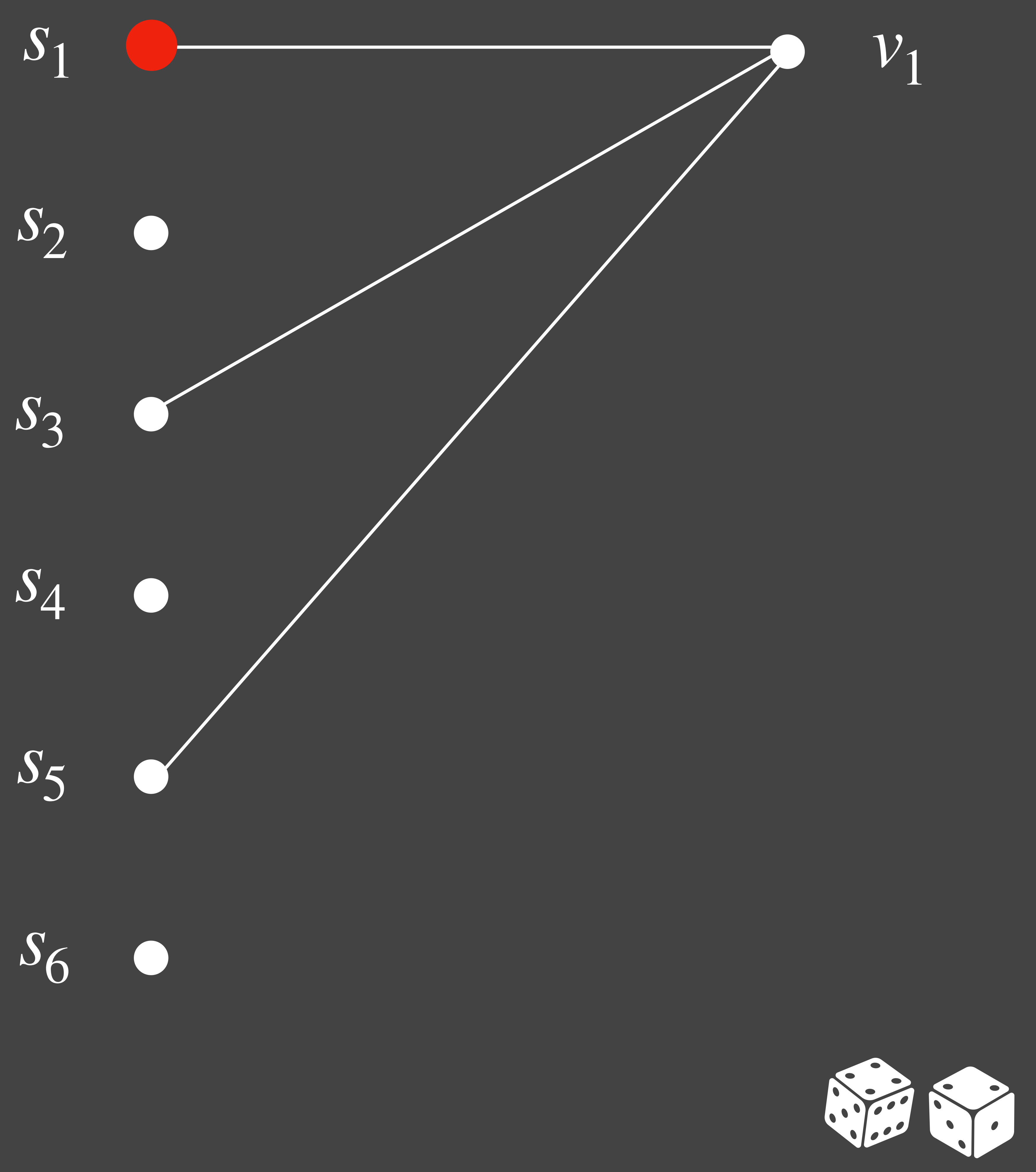
s_5 ●

s_6 ●

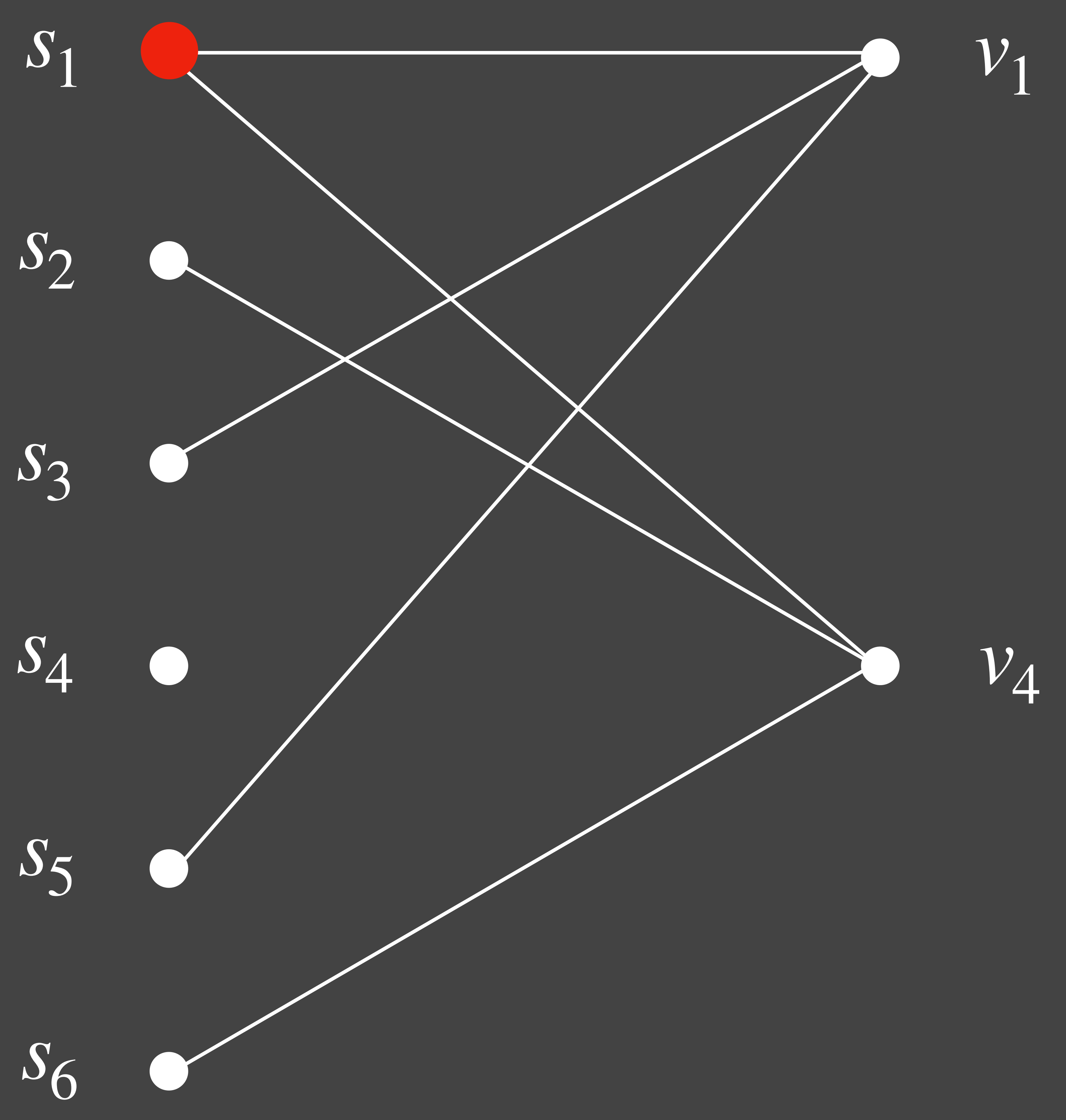
Relaxation 1: Random Order (RO)



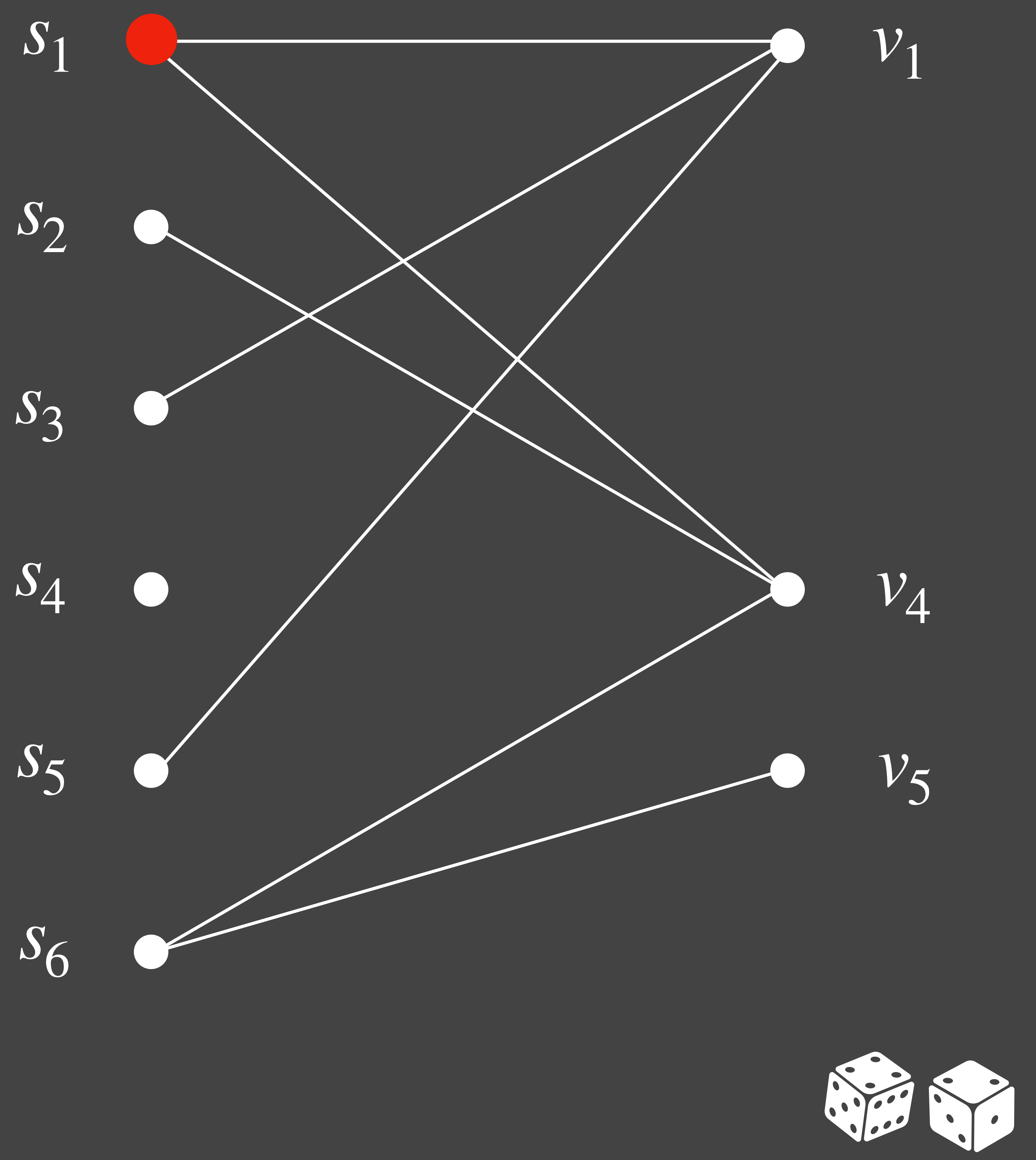
Relaxation 1: Random Order (RO)



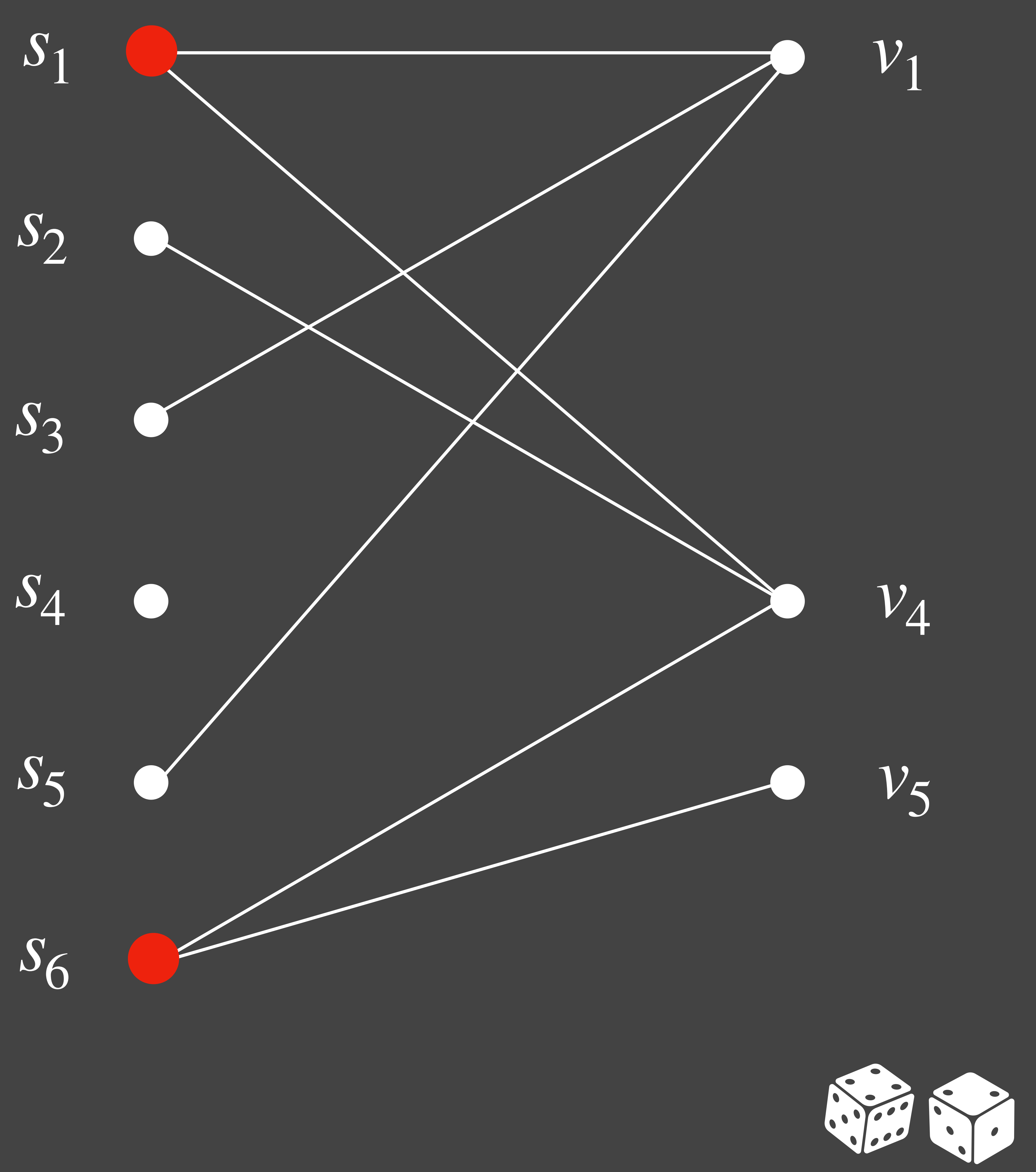
Relaxation 1: Random Order (RO)



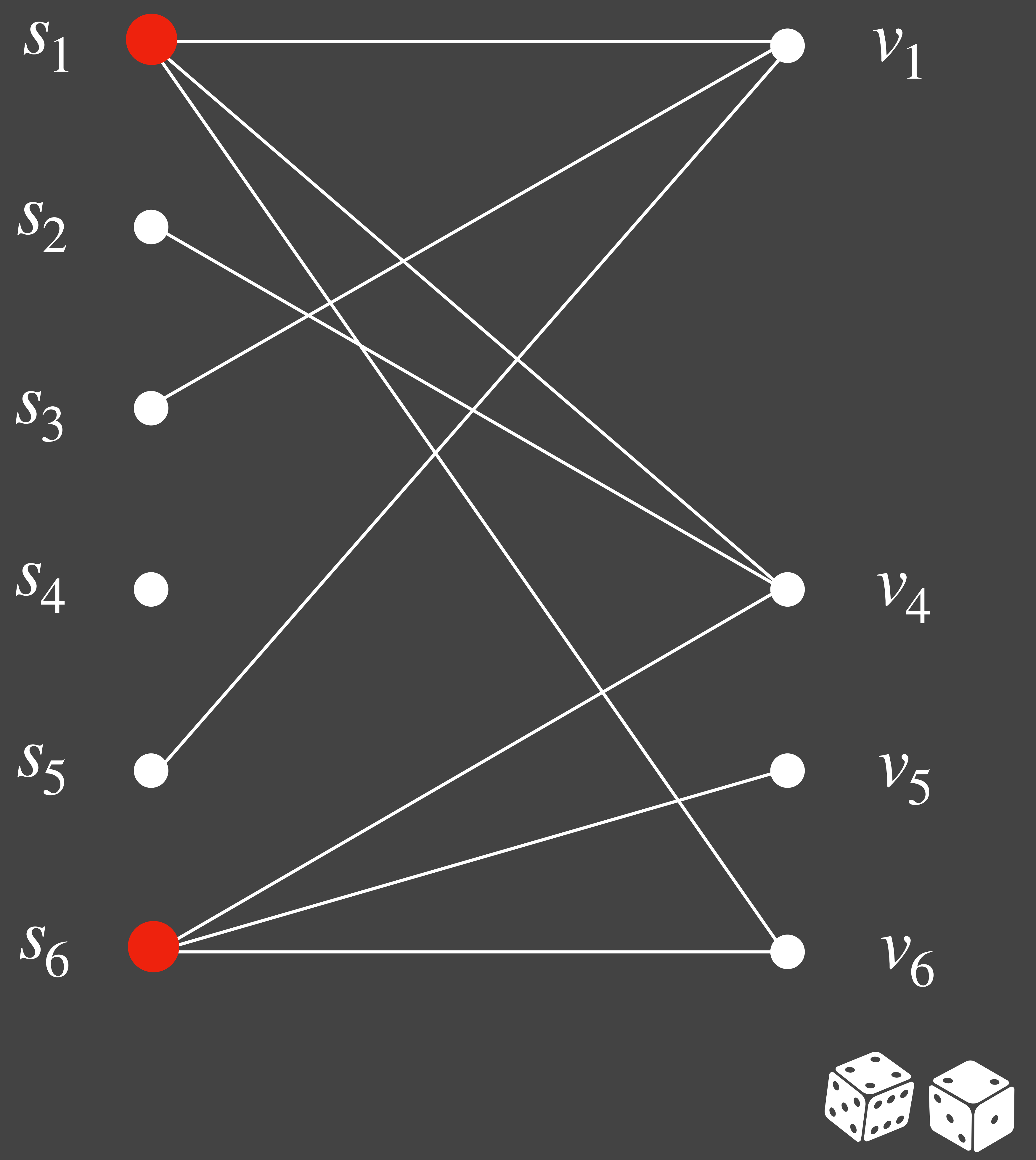
Relaxation 1: Random Order (RO)



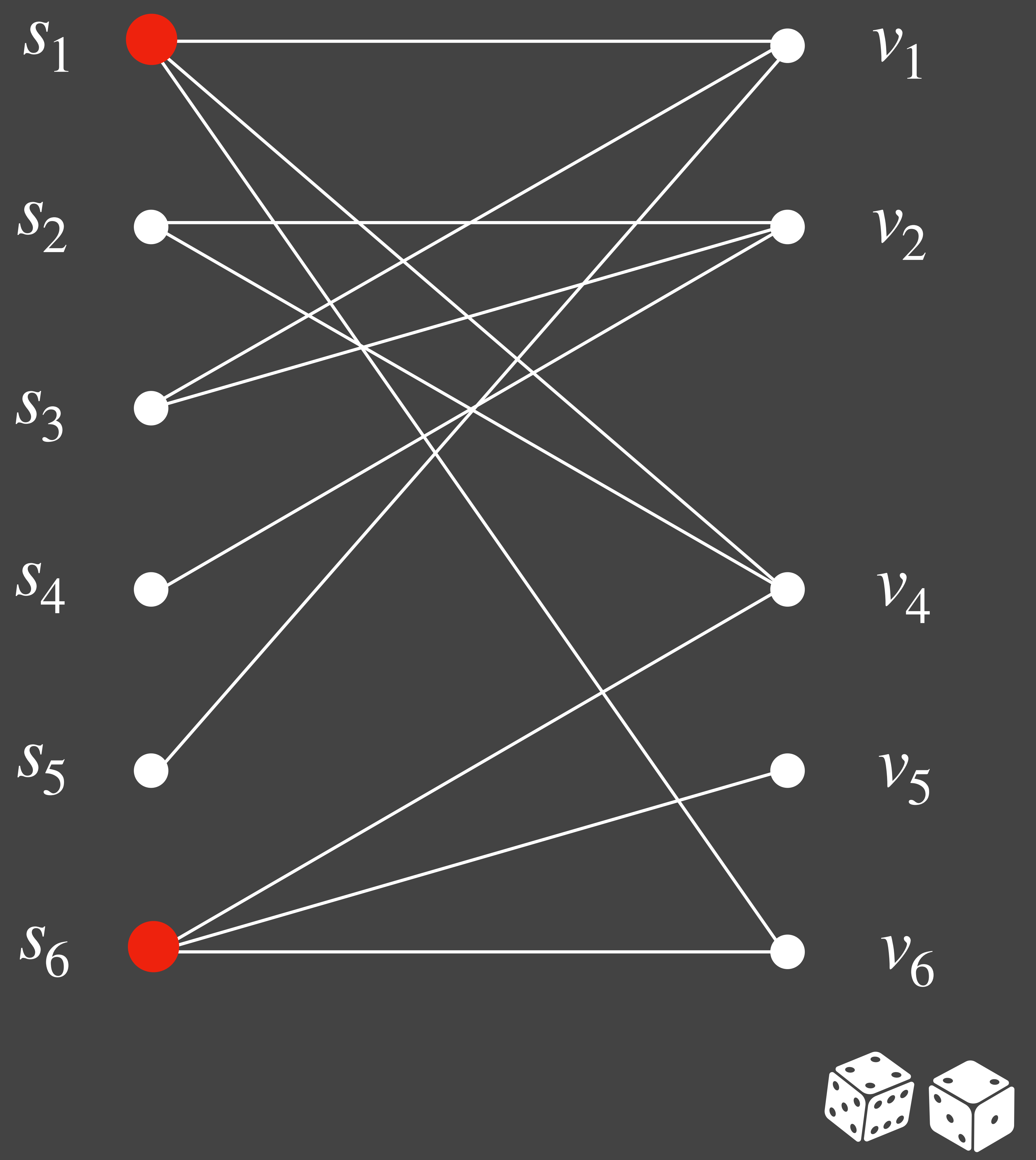
Relaxation 1: Random Order (RO)



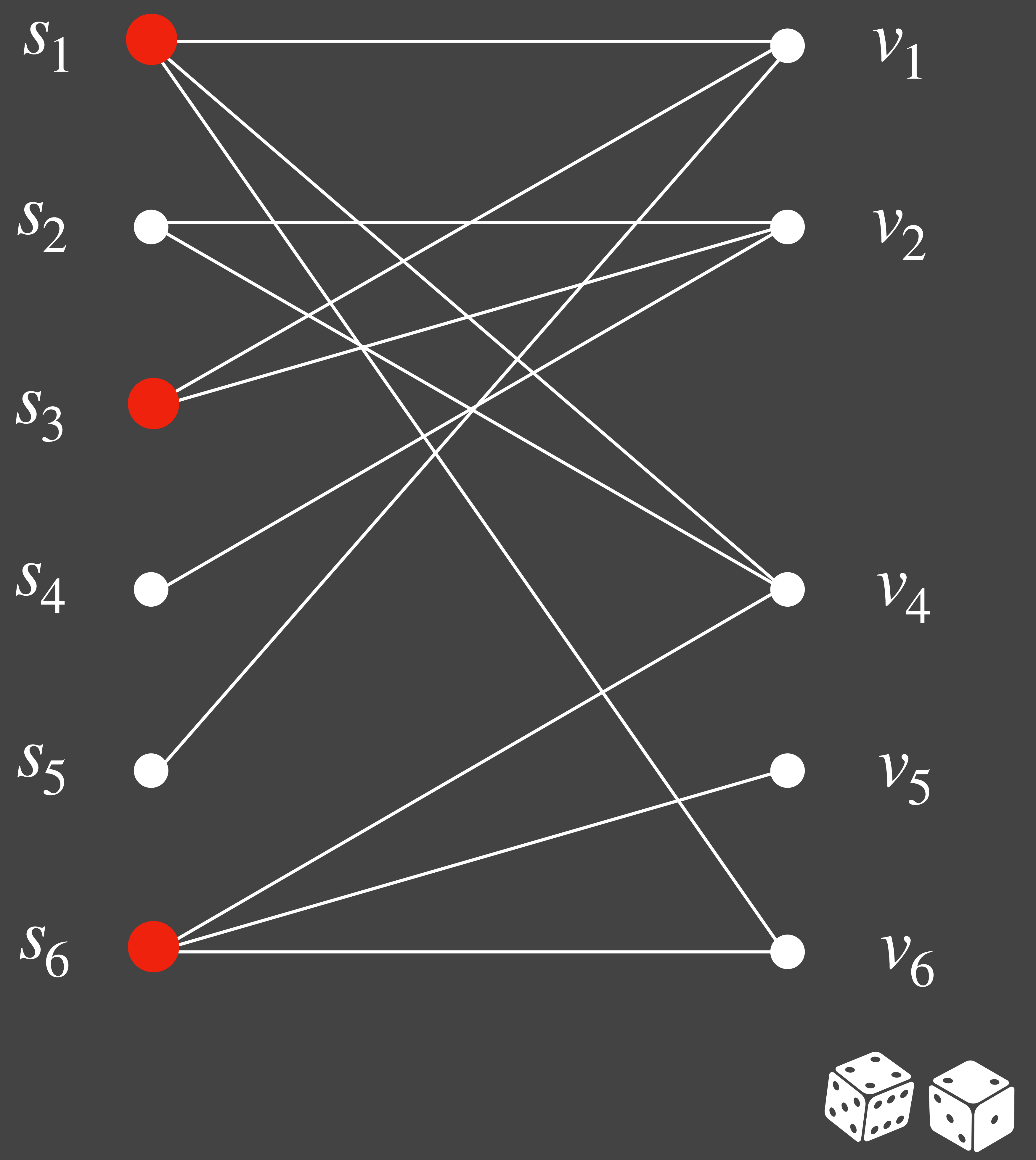
Relaxation 1: Random Order (RO)



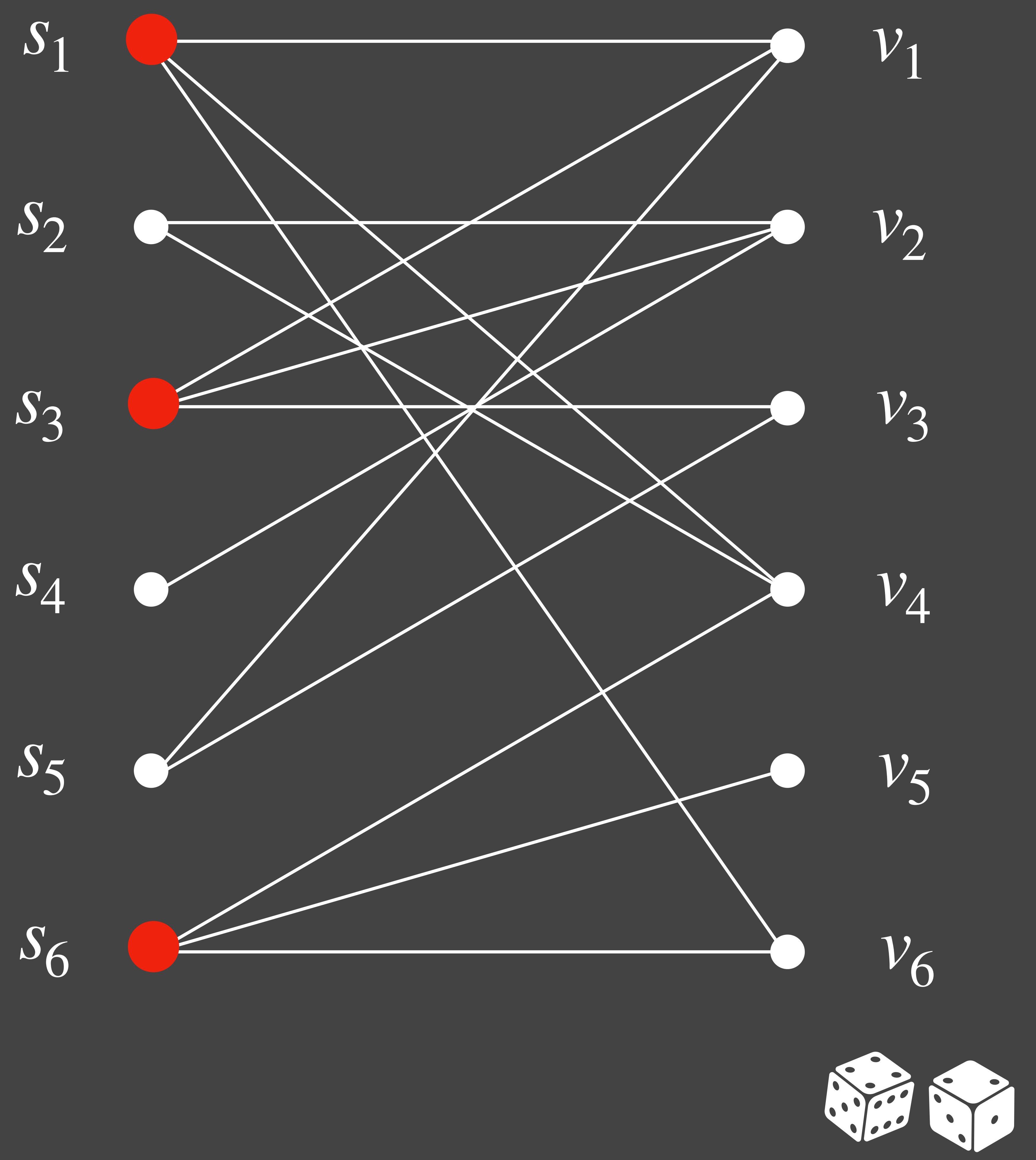
Relaxation 1: Random Order (RO)



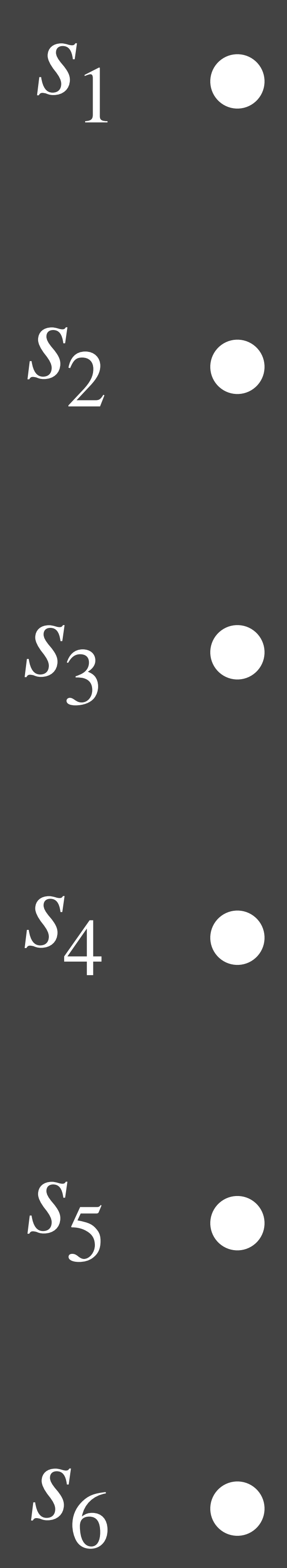
Relaxation 1: Random Order (RO)



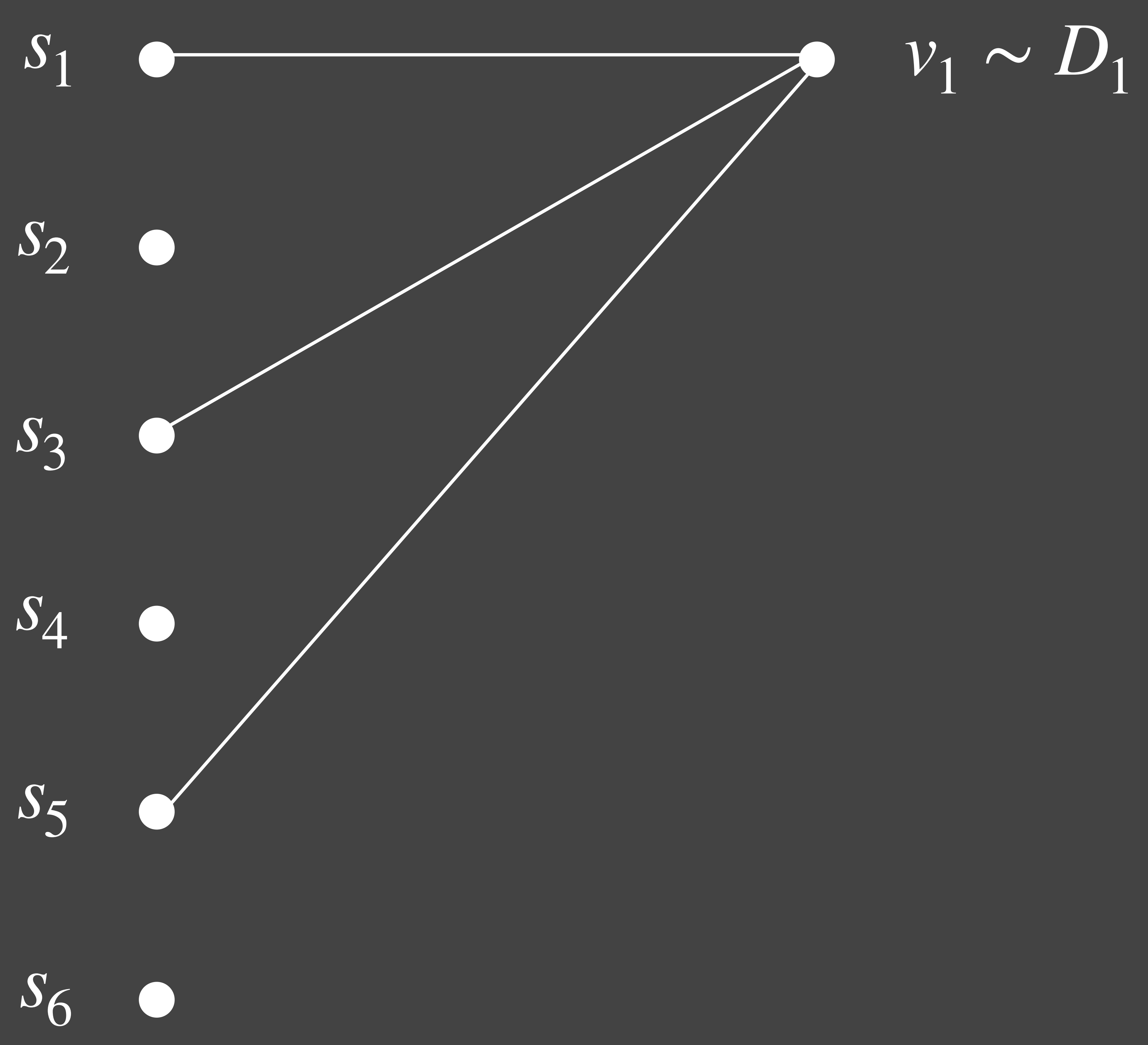
Relaxation 1: Random Order (RO)



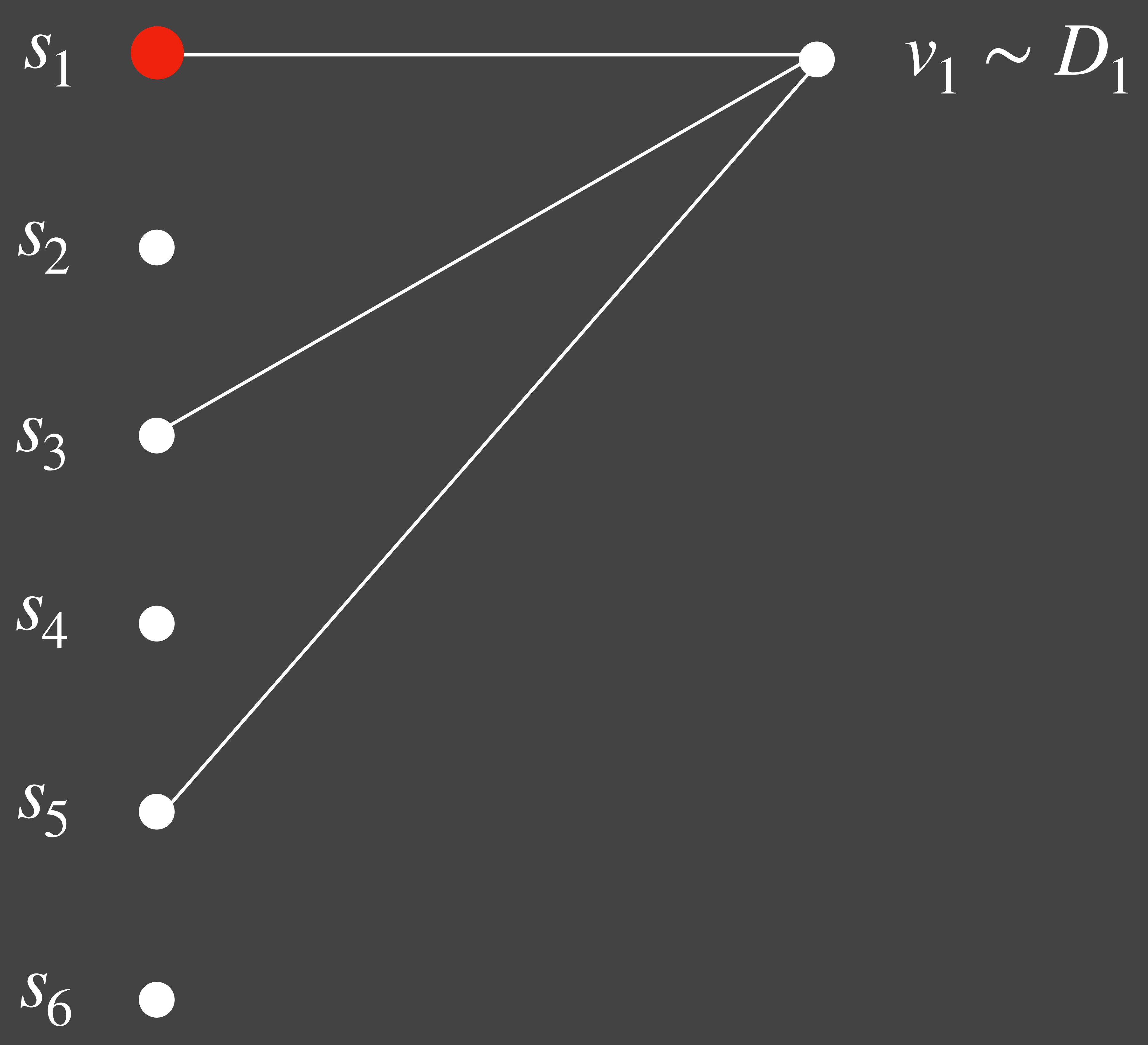
Relaxation 2: Random Instance



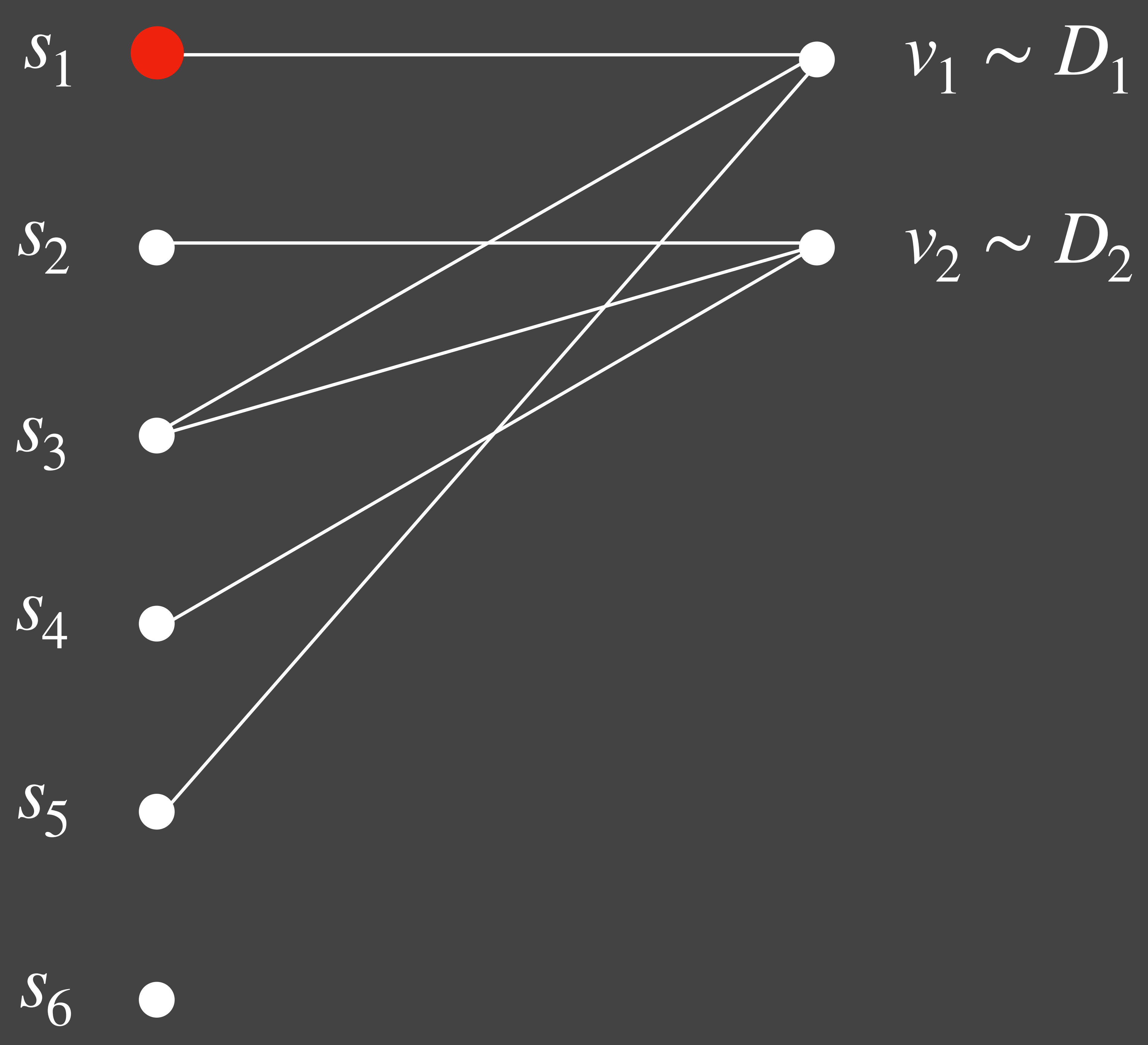
Relaxation 2: Random Instance



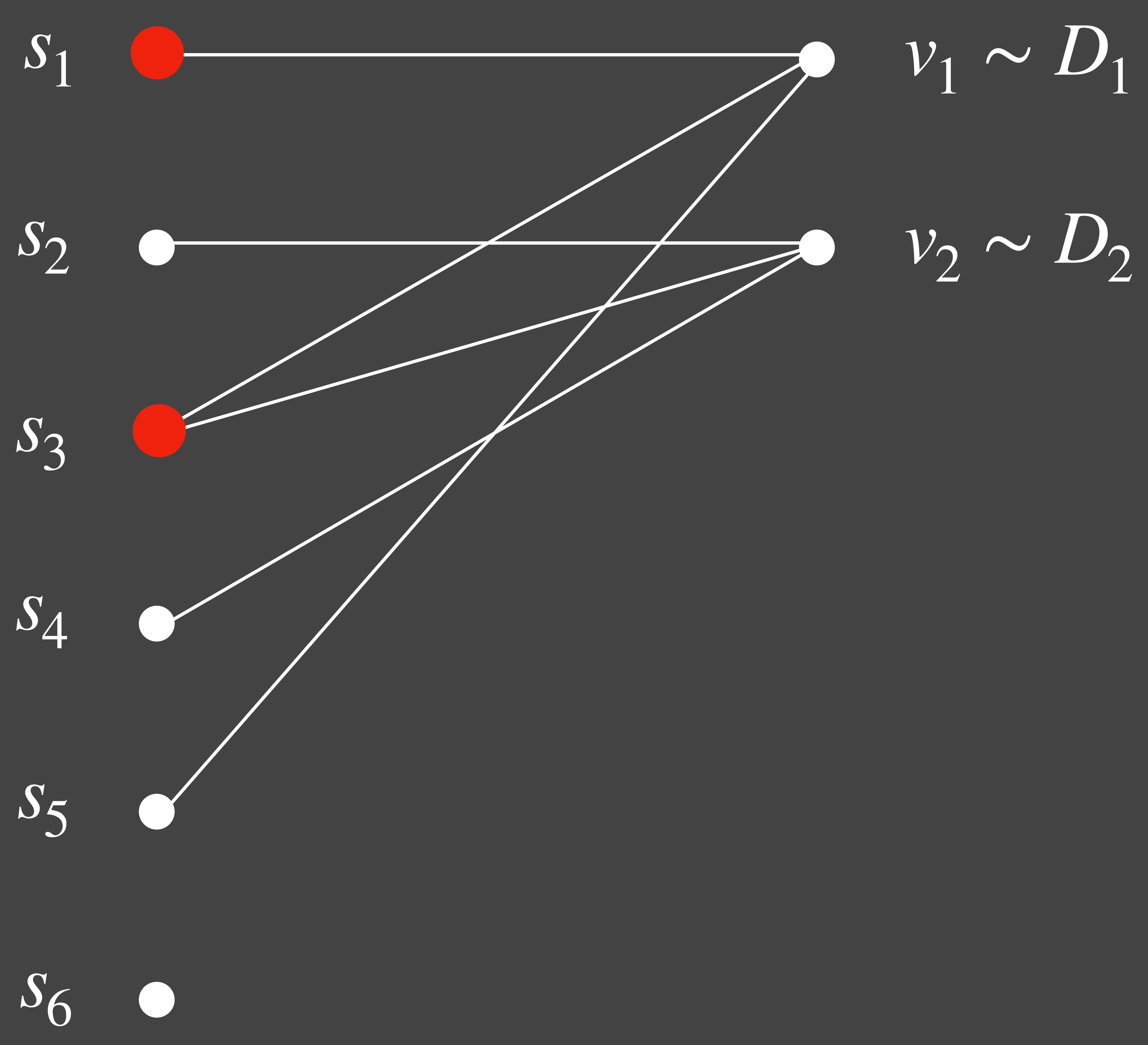
Relaxation 2: Random Instance



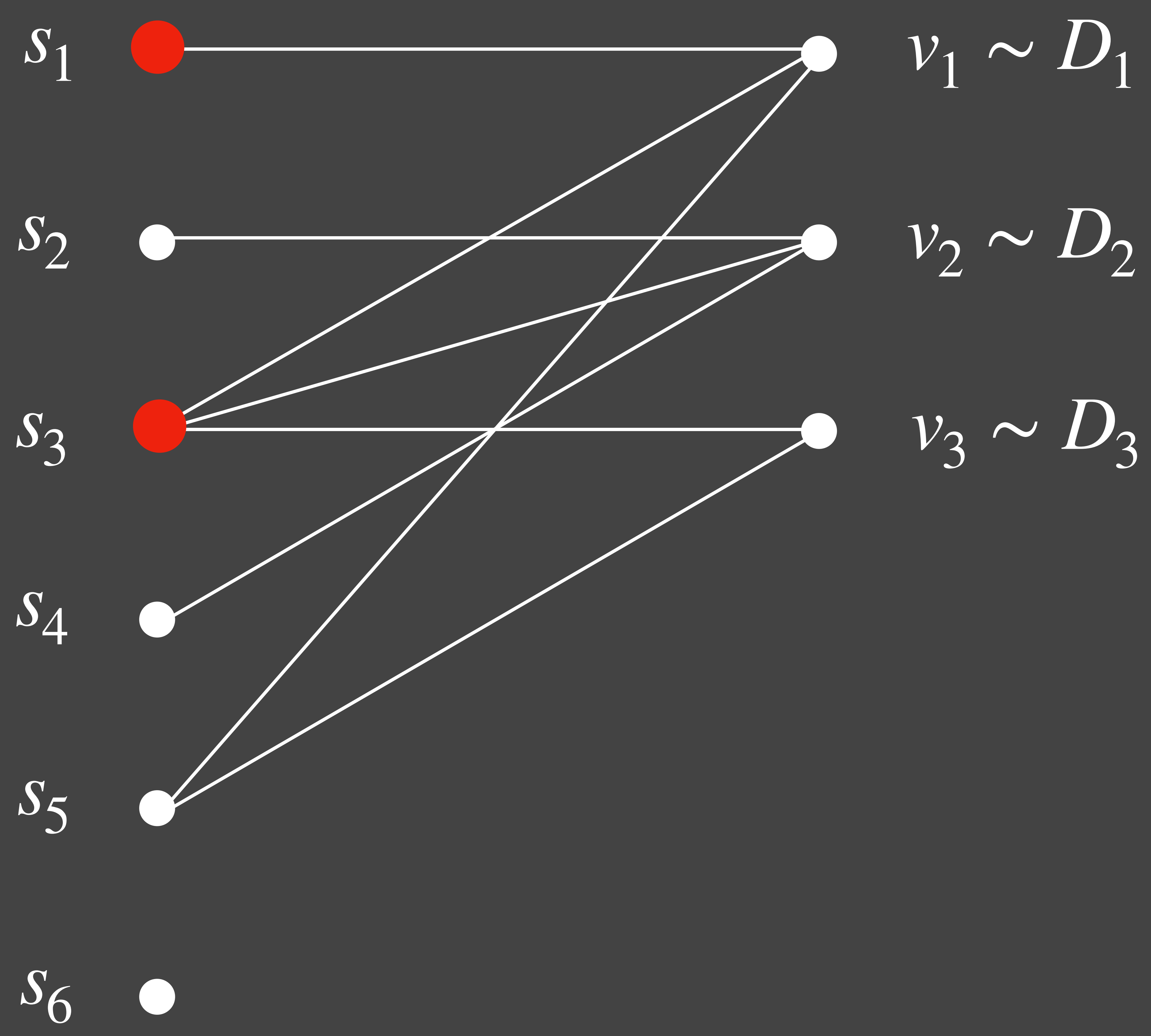
Relaxation 2: Random Instance



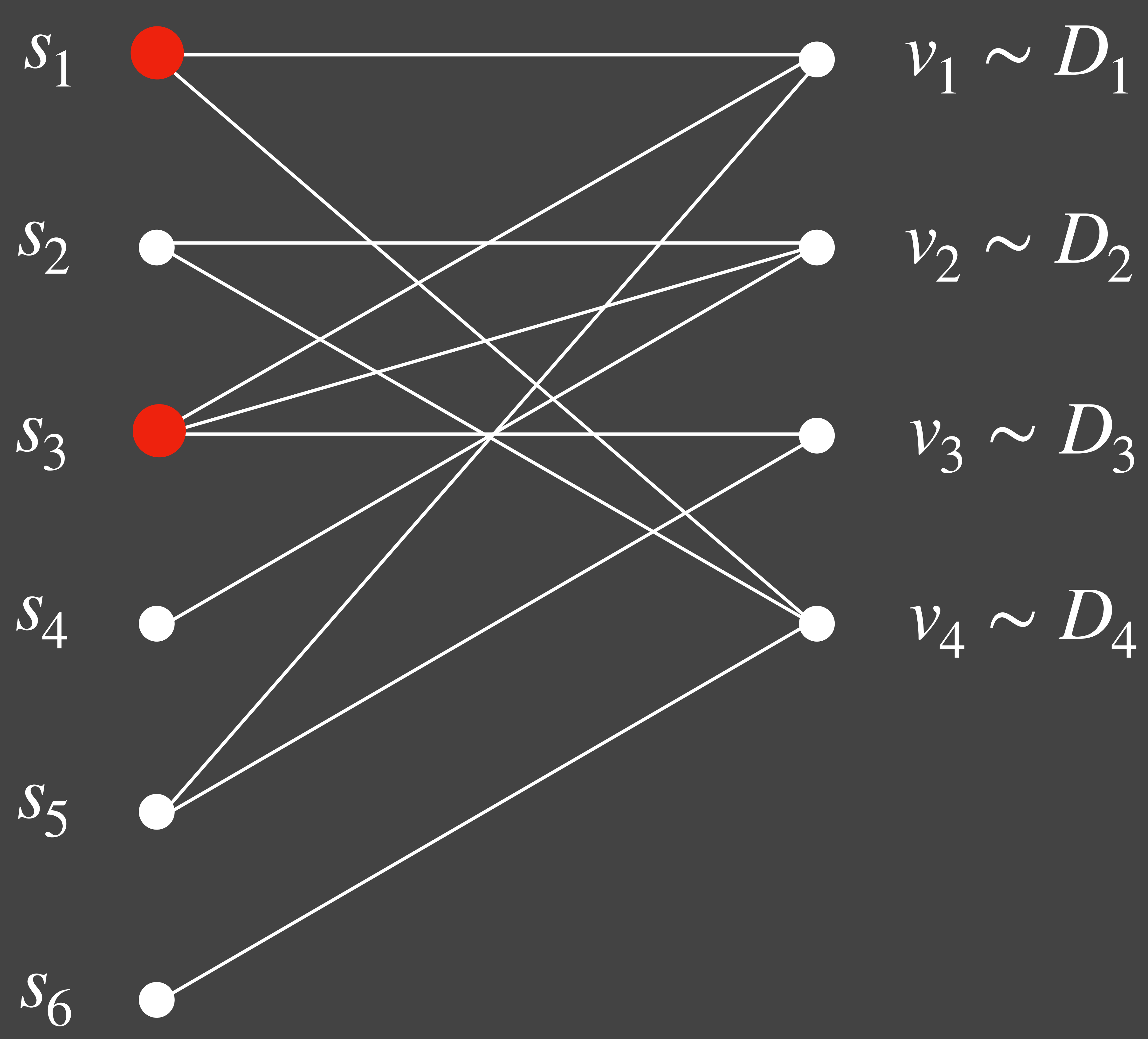
Relaxation 2: Random Instance



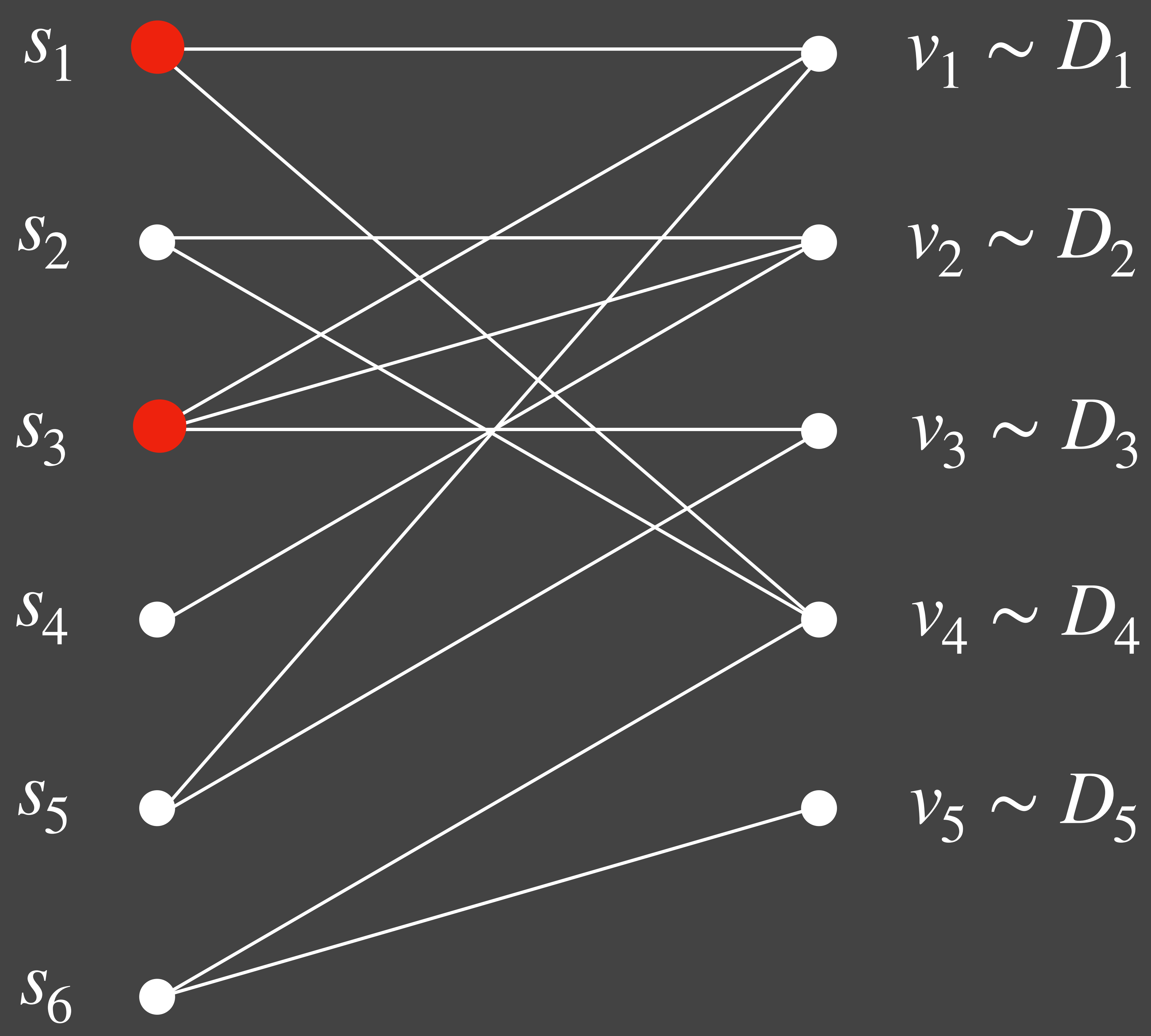
Relaxation 2: Random Instance



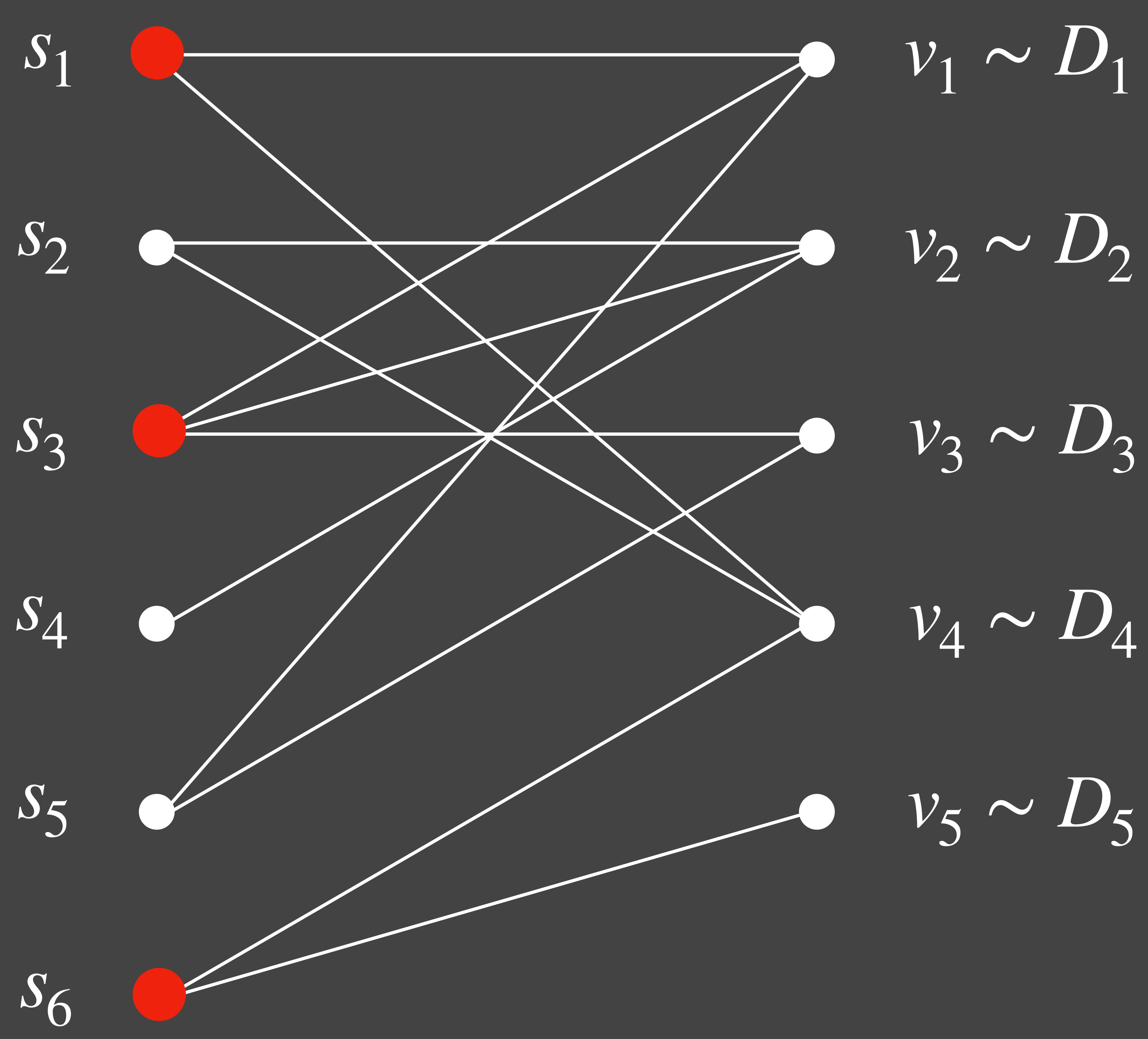
Relaxation 2: Random Instance



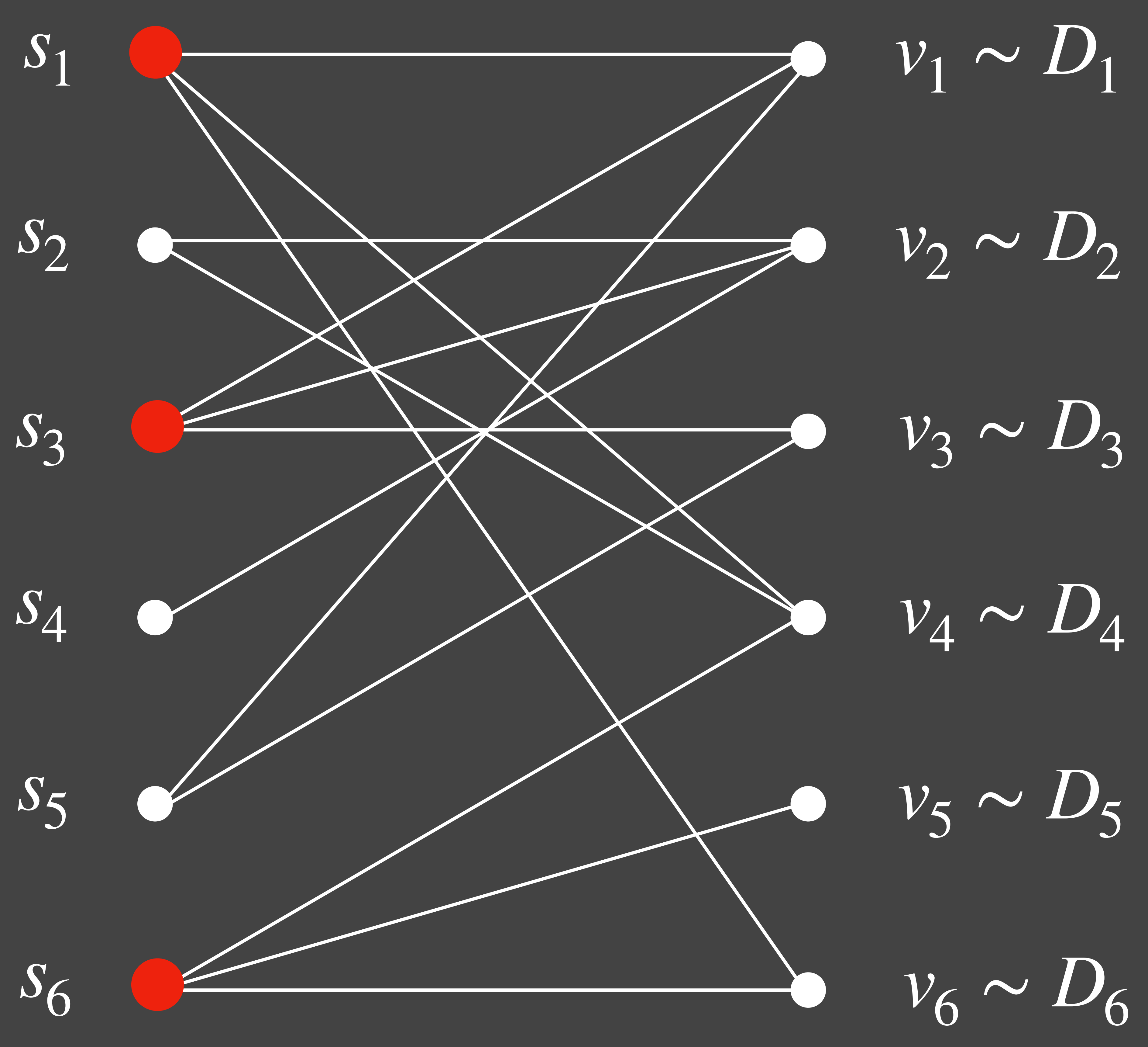
Relaxation 2: Random Instance



Relaxation 2: Random Instance



Relaxation 2: Random Instance



The Landscape

The Landscape

Instance

Arrival Order

	Random	Adversarial
Random		
Adversarial		$O(\log^2 n)$ [Alon+ 03] [Buchbinder Naor 09]

The Landscape

Arrival Order \ Instance	Random	Adversarial
Random	O(log(n [support size])) [Gupta Grandoni Leonardi Miettinen Sankowski Singh 08]	
Adversarial		$O(\log^2 n)$ [Alon+ 03] [Buchbinder Naor 09]

The Landscape

Instance

Arrival Order

	Random	Adversarial
Random	$O(\log(n\ [\text{support size}]))$ [Gupta Grandoni Leonardi Miettinen Sankowski Singh 08]	Secretary
Adversarial		$O(\log^2 n)$ [Alon+ 03] [Buchbinder Naor 09]

The Landscape

Instance

Arrival Order	Random	Adversarial
Random	O(log(n [support size])) [Gupta Grandoni Leonardi Miettinen Sankowski Singh 08]	Secretary
Adversarial	Prophet	$O(\log^2 n)$ [Alon+ 03] [Buchbinder Naor 09]

The Landscape

Instance

Arrival Order	Random	Adversarial
Random	$O(\log(n \text{ [support size]}))$ [Gupta Grandoni Leonardi Miettinen Sankowski Singh 08]	Secretary
Adversarial	Prophet	$O(\log^2 n)$ [Alon+ 03] [Buchbinder Naor 09]

Was believed $O(\log^2 n)$ best possible [Gupta+ 09]…

The Landscape

Instance

Arrival Order	Random	Adversarial
Random	O(log(n [support size])) [Gupta Grandoni Leonardi Miettinen Sankowski Singh 08]	O(log n) Our work
Adversarial		O(log² n) [Alon+ 03] [Buchbinder Naor 09]

Secretary

Prophet

Theorem [Gupta Kehne L. FOCS 21]:

Polynomial time algo for secretary Covering IP with approximation $O(\log n)$.

The Landscape

Instance

Arrival Order

	Random	Adversarial
Random	O(log(n [support size])) [Gupta Grandoni Leonardi Miettinen Sankowski Singh 08]	Secretary: O(log n) Our work
Adversarial	Prophet	O(log² n) [Alon+ 03] [Buchbinder Naor 09]

Theorem [Gupta Kehne L. FOCS 21]:

Polynomial time algo for secretary Covering IP with approximation $O(\log n)$.

New algorithm, LearnOrCover!
Not just new analysis of old algorithm.

The Landscape

Instance

Arrival Order	Random	Adversarial
Random	O(log(n [support size])) [Gupta Grandoni Leonardi Miettinen Sankowski Singh 08]	Secretary: O(log n) Our work
Adversarial	Prophet: O(log n) Our work	$O(\log^2 n)$ [Alon+ 03] [Buchbinder Naor 09]

Theorem [Gupta Kehne L. FOCS 21]:

Polynomial time algo for secretary Covering IP with approximation $O(\log n)$.

Theorem [Gupta Kehne L. In submission]:

Polynomial time algo for prophet Covering IPs with approximation $O(\log n)$.

The Landscape

Instance

Arrival Order	Random	Adversarial
Random	O(log(n [support size])) [Gupta Grandoni Leonardi Miettinen Sankowski Singh 08]	Secretary O(log n) Our work
Adversarial	O(log n) Our work Prophet	$O(\log^2 n)$ [Alon+ 03] [Buchbinder Naor 09]

Bonus!
Only need 1 sample from each D_i!

Theorem [Gupta Kehne L. FOCS 21]:

Polynomial time algo for secretary Covering IP with approximation $O(\log n)$.

Theorem [Gupta Kehne L. In submission]:

Polynomial time algo for prophet Covering IPs with approximation $O(\log n)$.

The Landscape

Bonus!
1-pass Streaming Algorithm!

Instance

Arrival Order	Random	Adversarial (Secretary)
Random	O(log(n [support size])) [Gupta Grandoni Leonardi Miettinen Sankowski Singh 08]	O(log n) Our work
Adversarial (Prophet)	O(log n) Our work	$O(\log^2 n)$ [Alon+ 03] [Buchbinder Naor 09]

Bonus!
Only need 1 sample from each D_i!

Theorem [Gupta Kehne L. FOCS 21]:

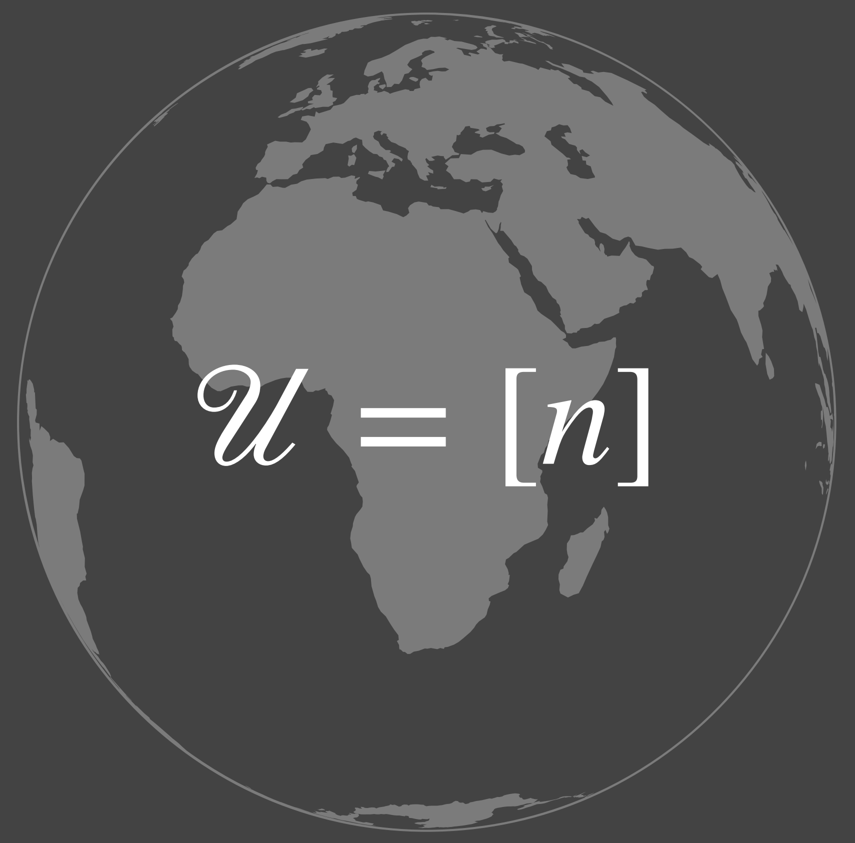
Polynomial time algo for secretary Covering IP with approximation $O(\log n)$.

Theorem [Gupta Kehne L. In submission]:

Polynomial time algo for prophet Covering IPs with approximation $O(\log n)$.

LearnOrCover

LearnOrCover

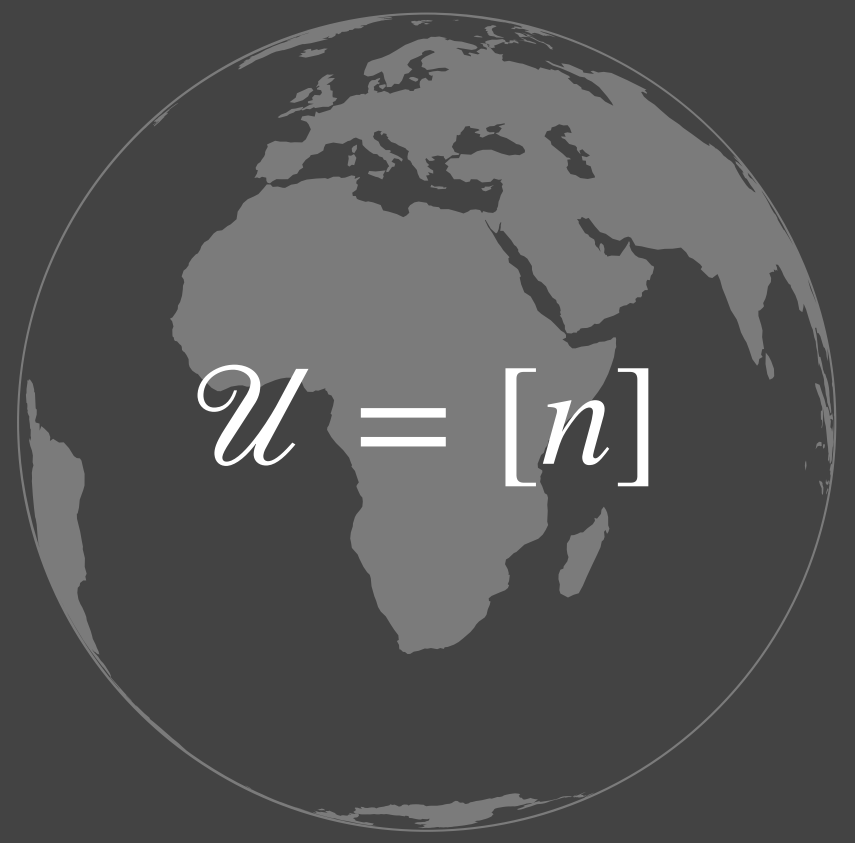


$$\mathcal{U} = [n]$$

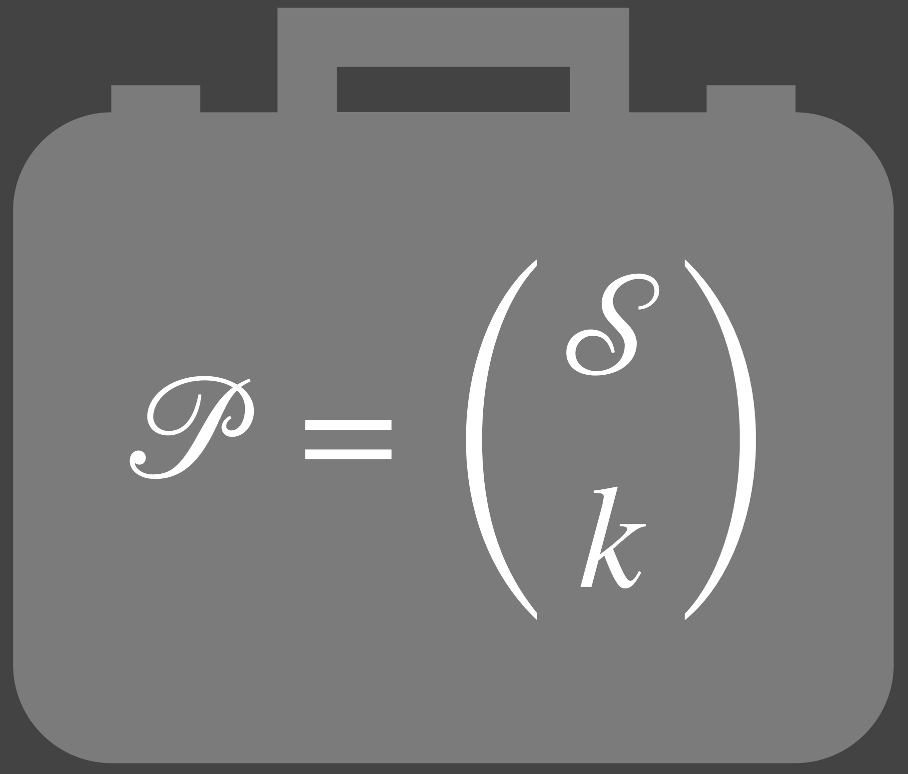
$$\mathcal{P} = \binom{\mathcal{S}}{k}$$

$$k := |OPT|$$

LearnOrCover



$\mathcal{U} = [n]$

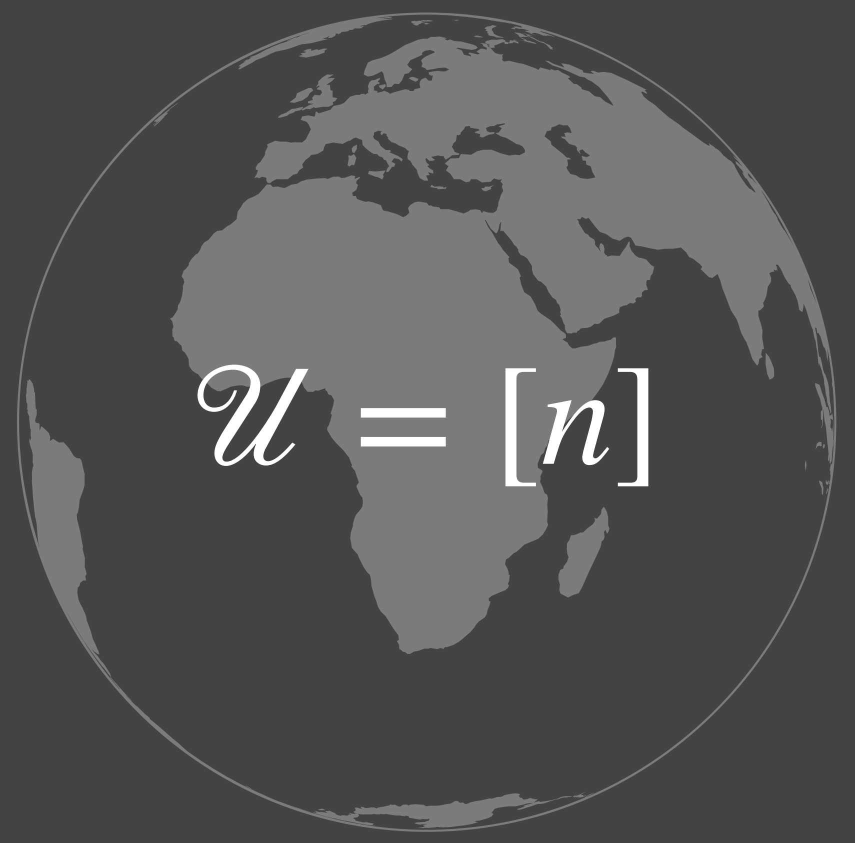


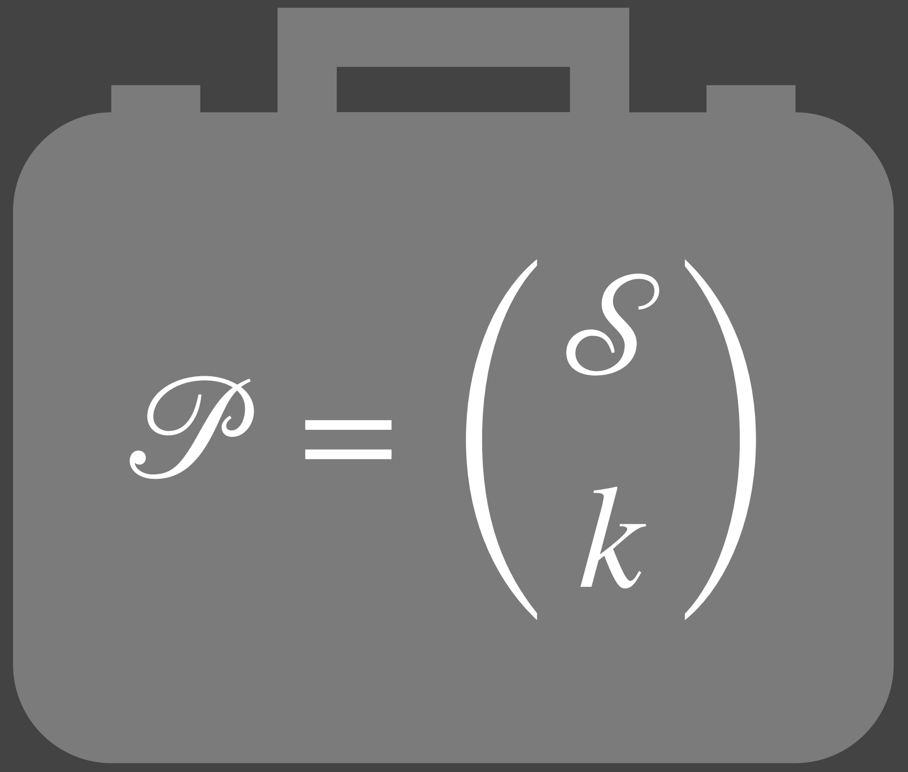
$\mathcal{P} = \binom{\mathcal{S}}{k}$

$k := |OPT|$

@ time t, element v arrives:

LearnOrCover



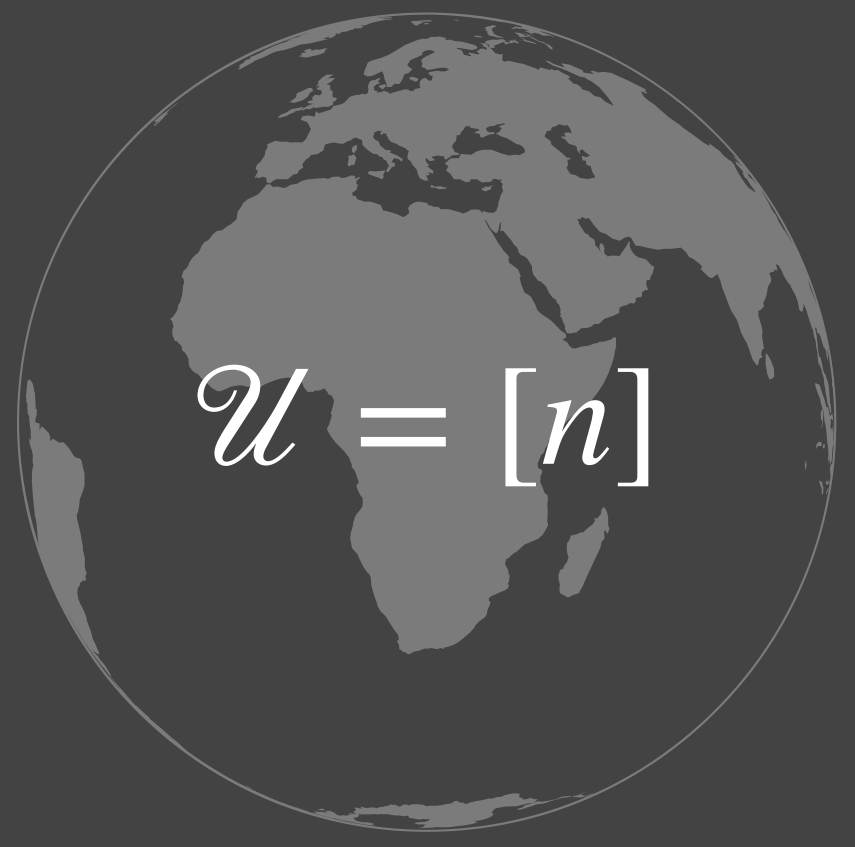


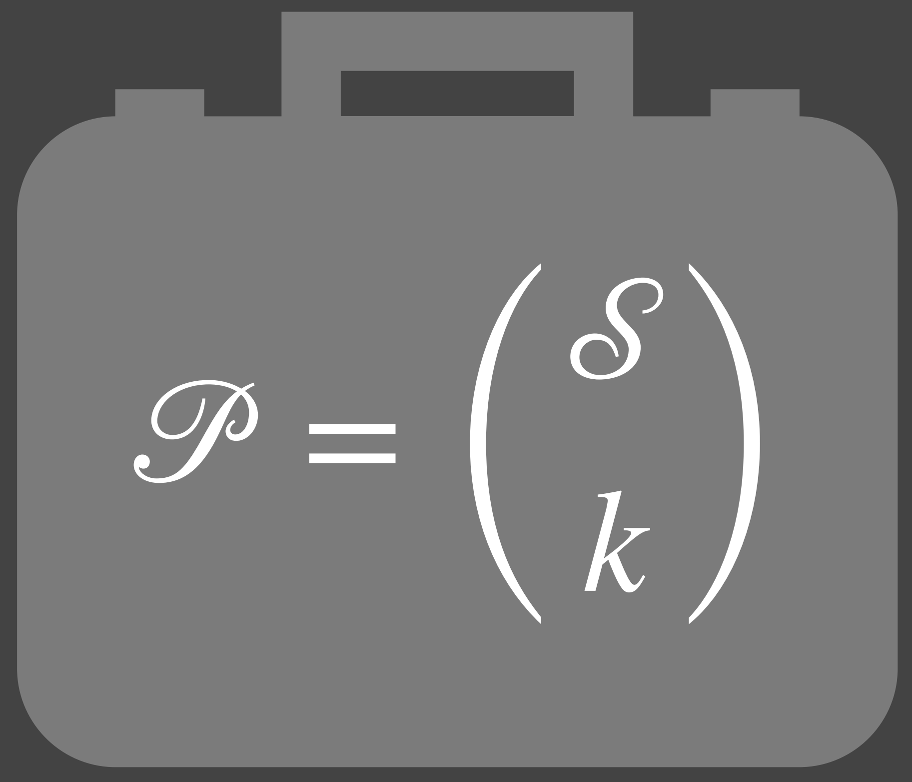
$k := |OPT|$

@ time t, element v arrives:

If v covered, do nothing.

LearnOrCover





$k := |OPT|$

@ time t, element v arrives:

If v covered, do nothing.

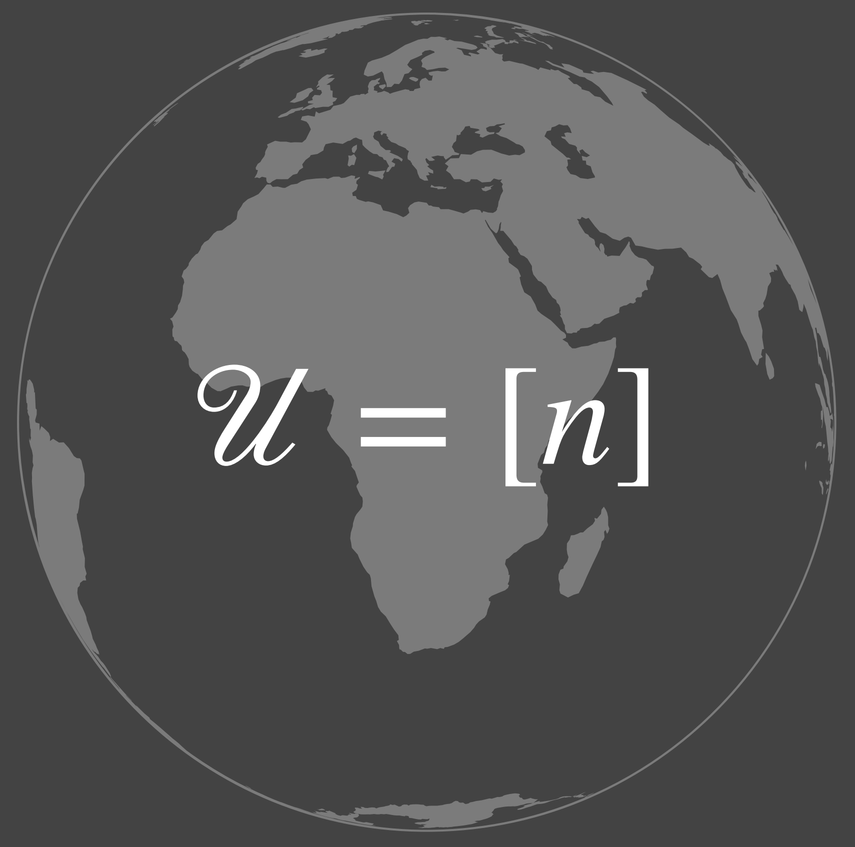
Else:

(I) Buy random set R from $\mathscr{P}$ to cover v.

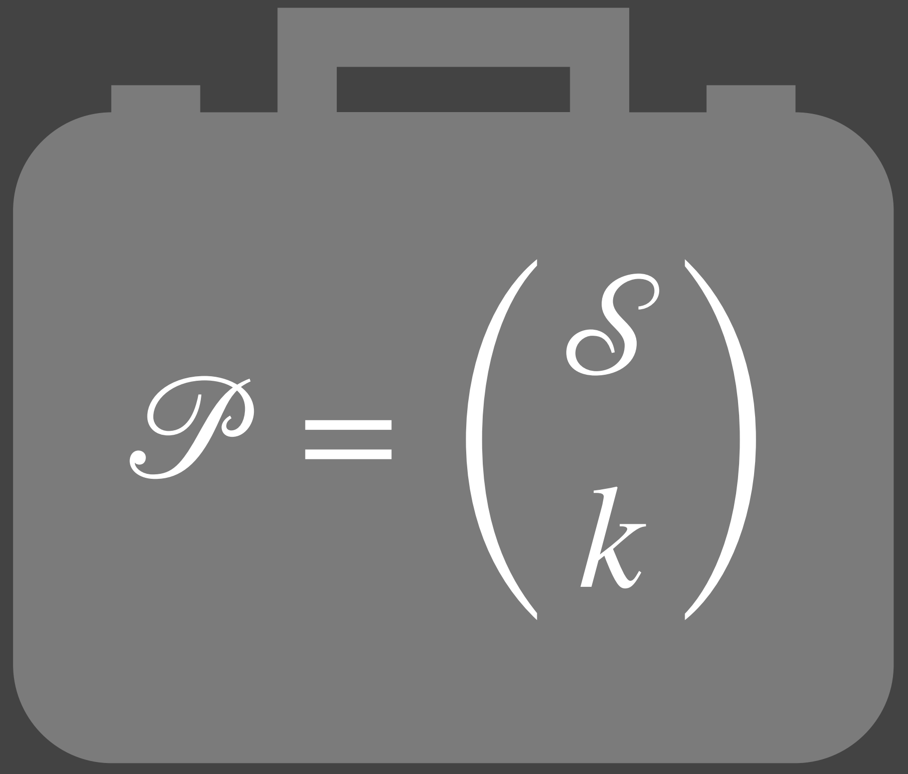
(II) “Prune” $P \not\ni v$ from $\mathscr{P}$.

LearnOrCover

Proof idea: progress learning or covering.



$\mathcal{U} = [n]$



$\mathscr{P} = \binom{\mathcal{S}}{k}$

$k := |OPT|$

@ time t, element v arrives:

If v covered, do nothing.

Else:

(I) Buy random set R from $\mathscr{P}$ to cover v.

(II) “Prune” $P \not\ni v$ from $\mathscr{P}$.

LearnOrCover

Proof idea: progress learning or covering.

$\mathscr{U} = [n]$

$\mathscr{P} = \binom{\mathscr{S}}{k}$

$k := |OPT|$

s_1

s_2

s_3

s_4

s_5

s_6

@ time t, element v arrives:

If v covered, do nothing.

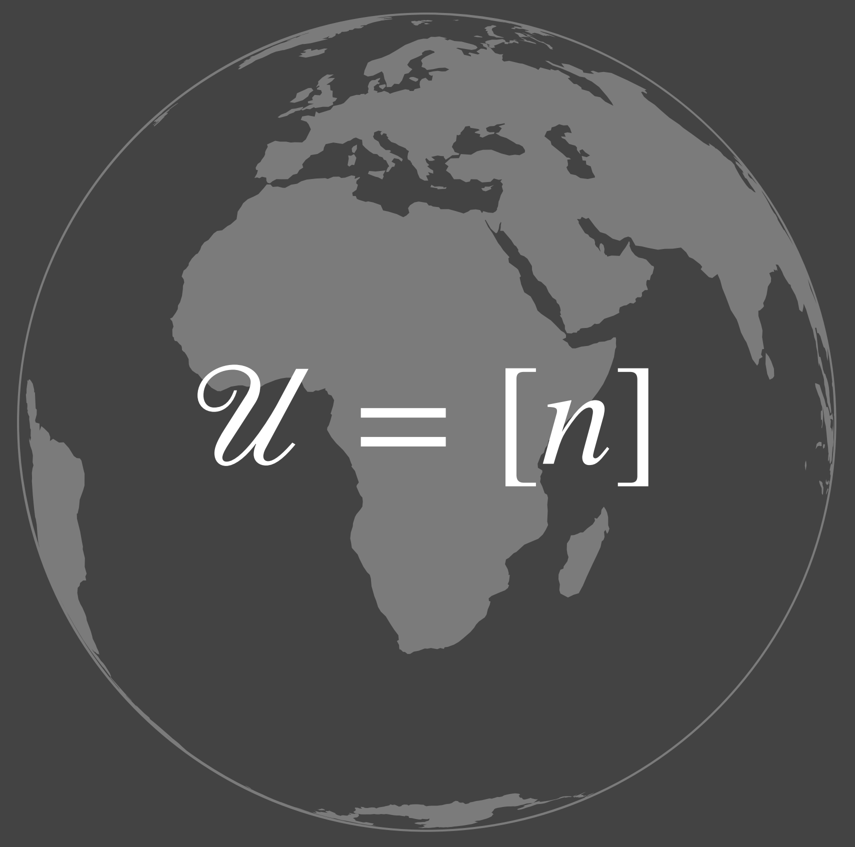
Else:

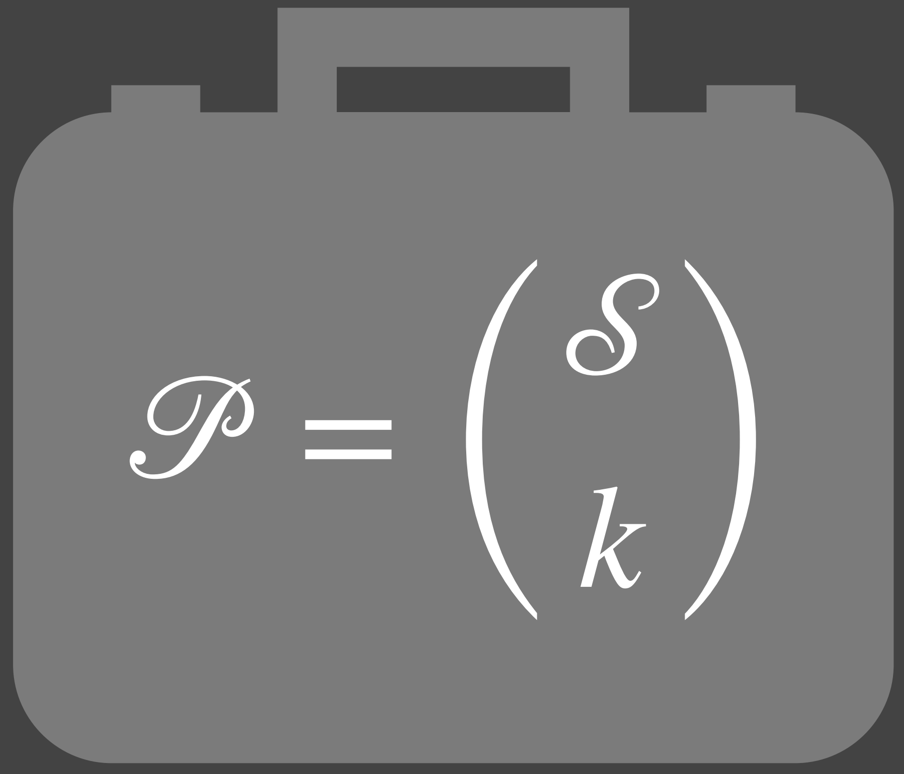
(I) Buy random set R from $\mathscr{P}$ to cover v.

(II) "Prune" $P \not\ni v$ from $\mathscr{P}$.

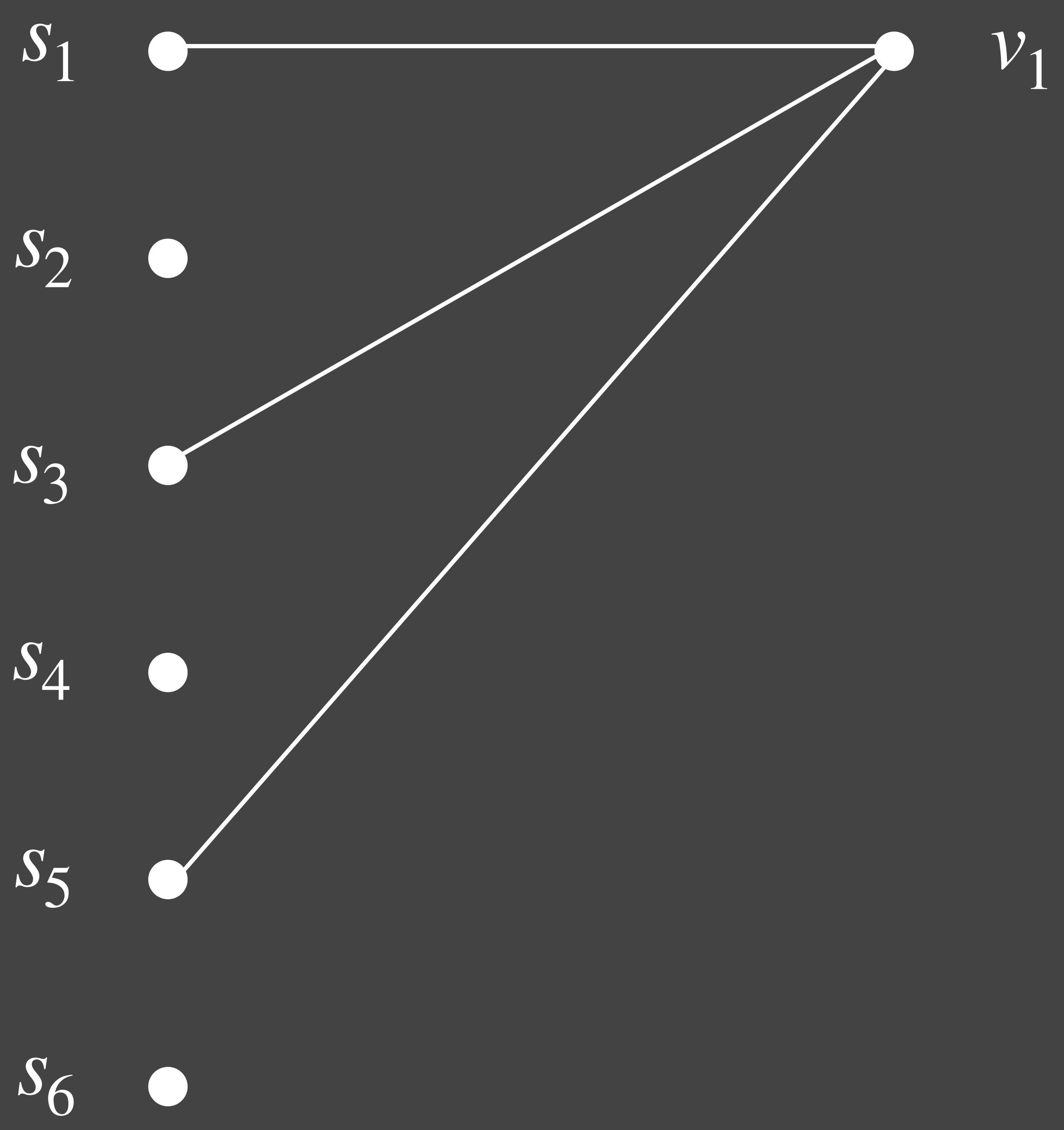
LearnOrCover

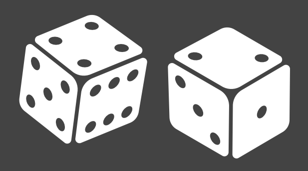
Proof idea: progress learning or covering.





$k := |OPT|$





@ time t, element v arrives:

If v covered, do nothing.

Else:

(I) Buy random set R from $\mathscr{P}$ to cover v.

(II) "Prune" $P \not\ni v$ from $\mathscr{P}$.

LearnOrCover

$\mathscr{U} = [n]$

$\mathscr{P} = \binom{\mathscr{S}}{k}$

$k := |OPT|$

@ time t, element v arrives:

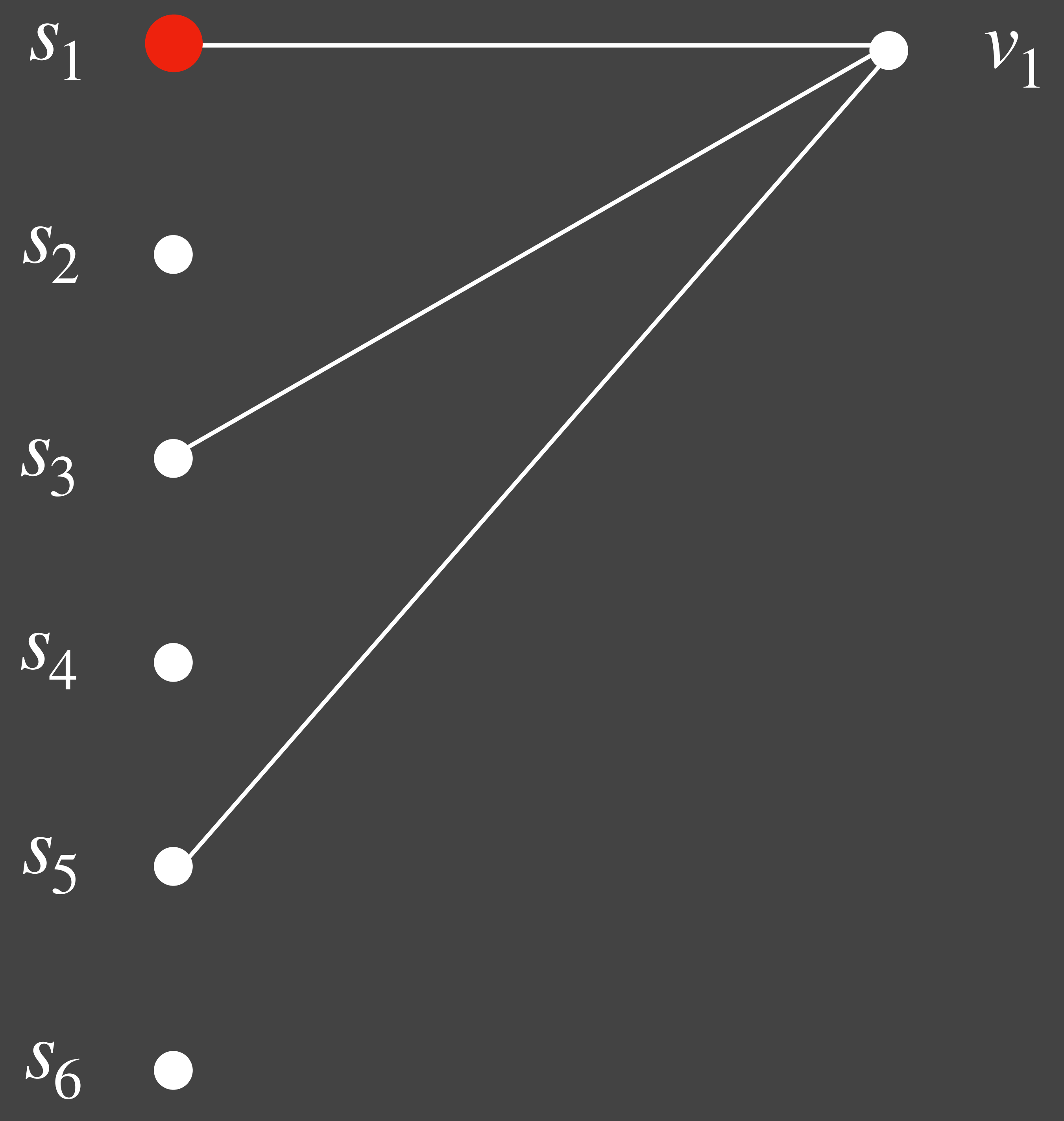
If v covered, do nothing.

Else:

(I) Buy random set R from $\mathscr{P}$ to cover v.

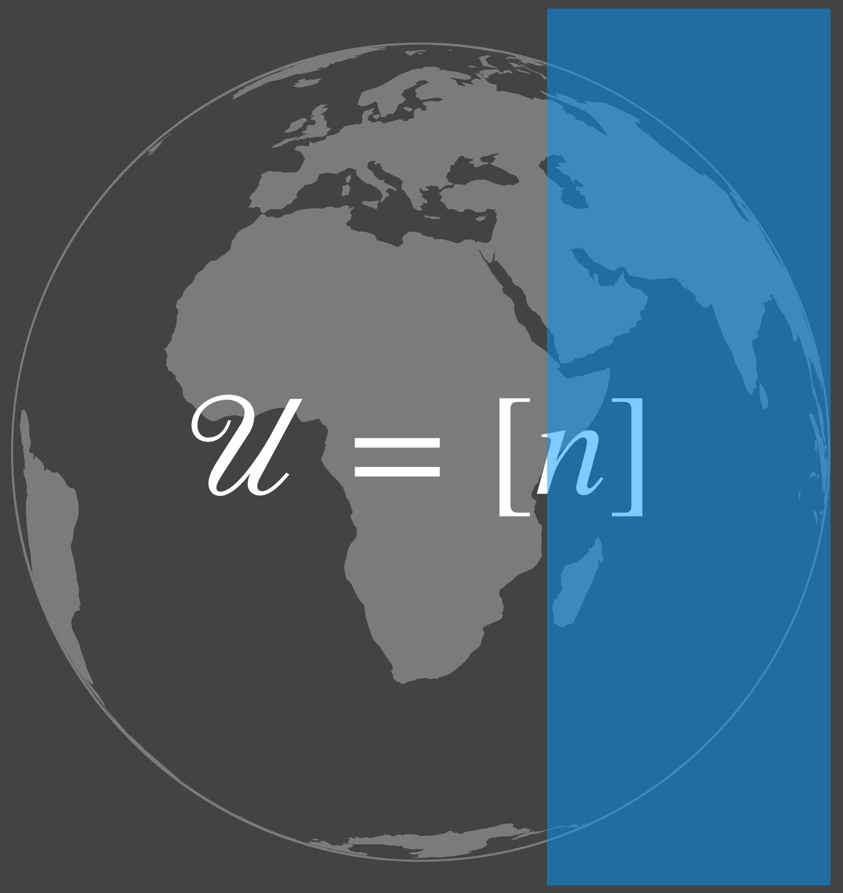
(II) "Prune" $P \not\ni v$ from $\mathscr{P}$.

Proof idea: progress learning or covering.

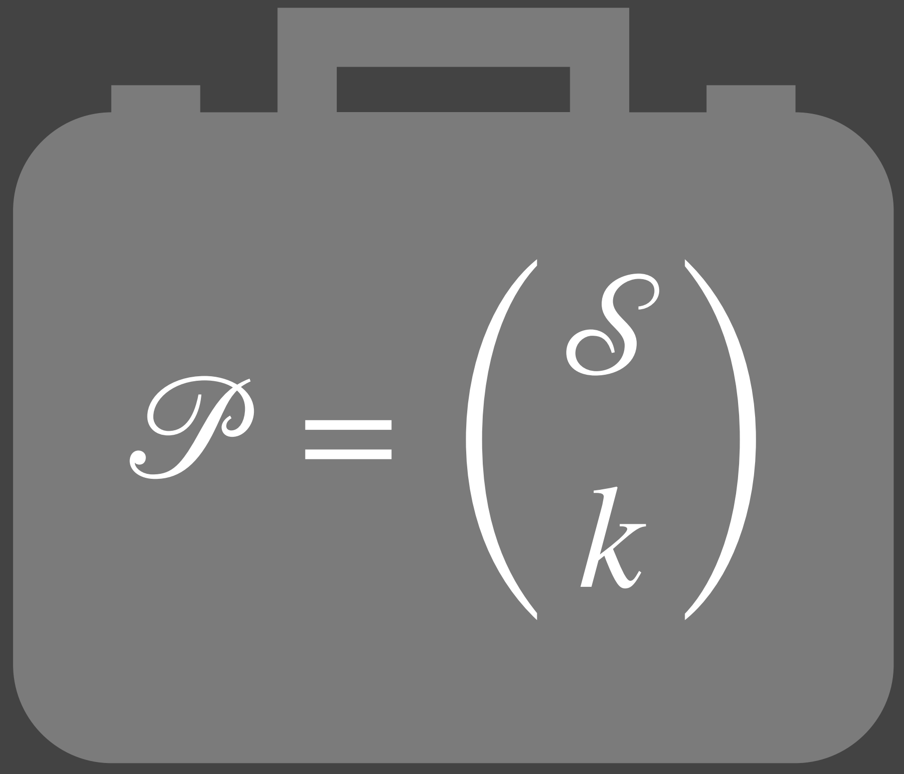


LearnOrCover

Proof idea: progress learning or covering.

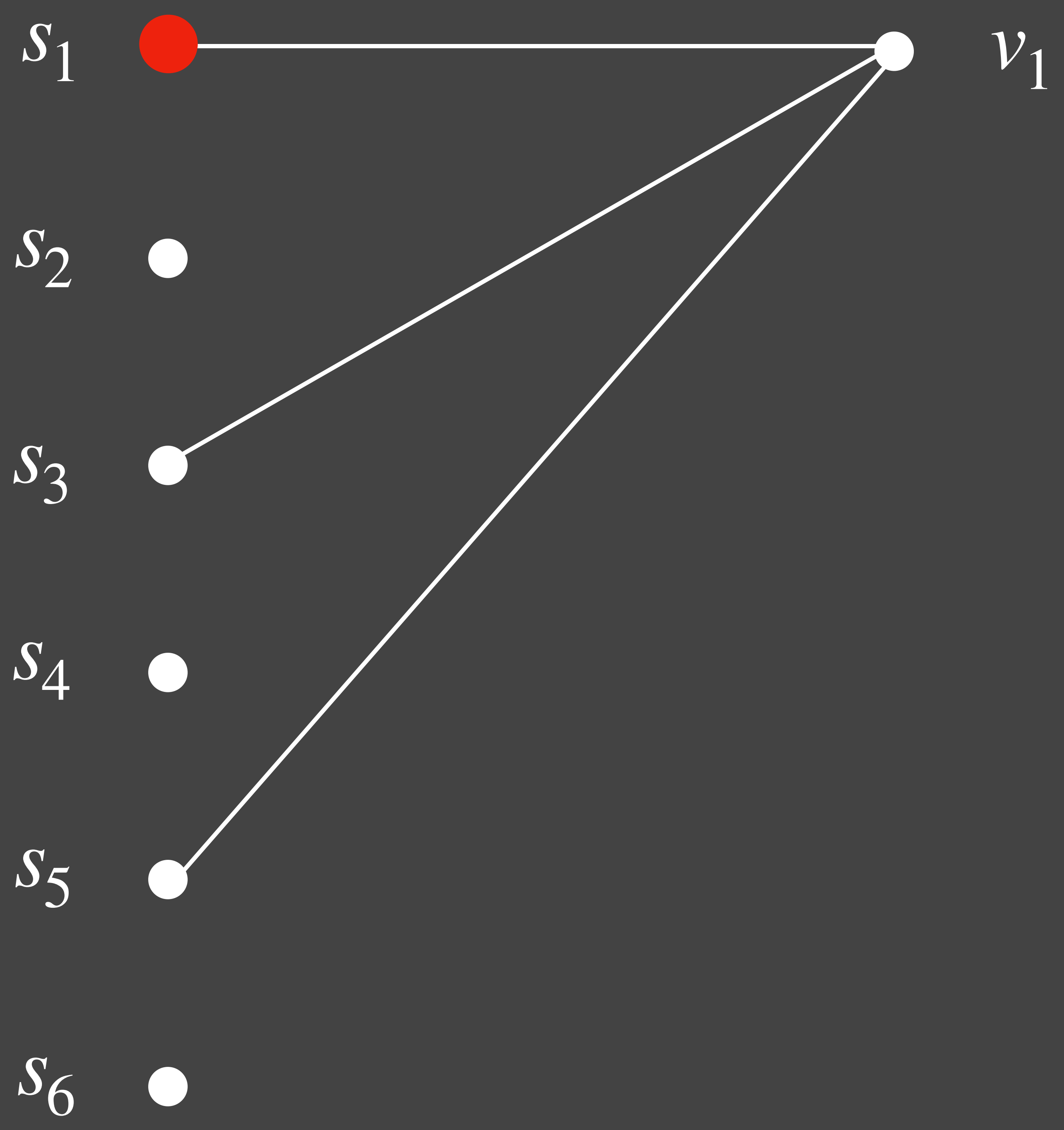


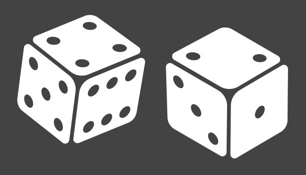
$\mathscr{U} = [n]$



$\mathscr{P} = \binom{\mathscr{S}}{k}$

$k := |OPT|$





@ time t, element v arrives:

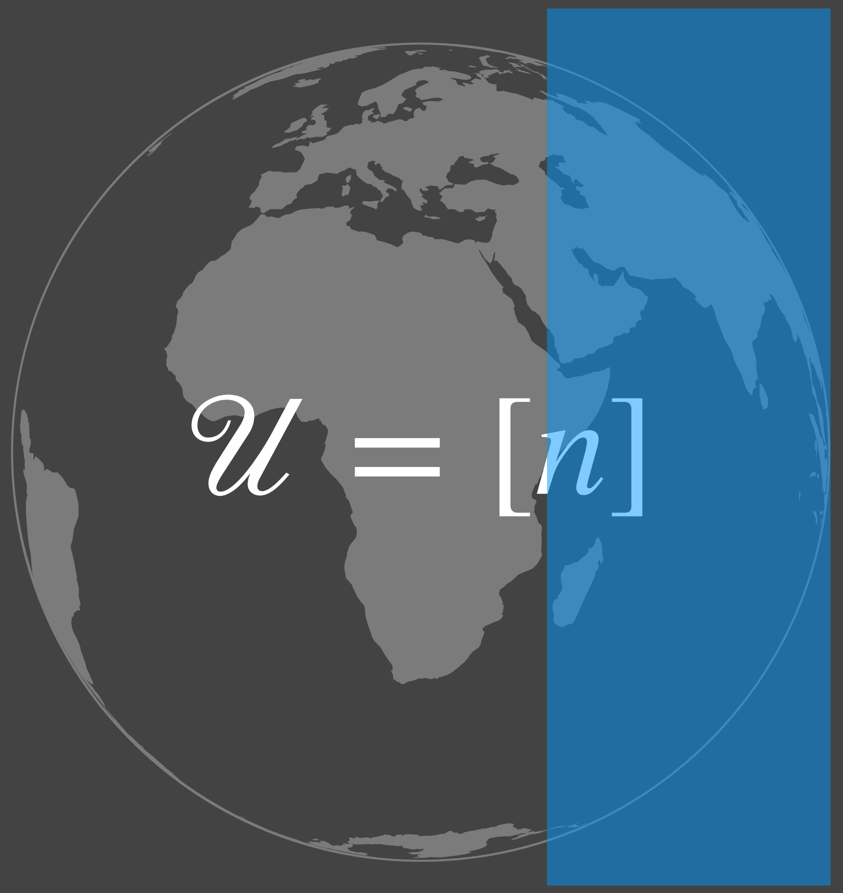
If v covered, do nothing.

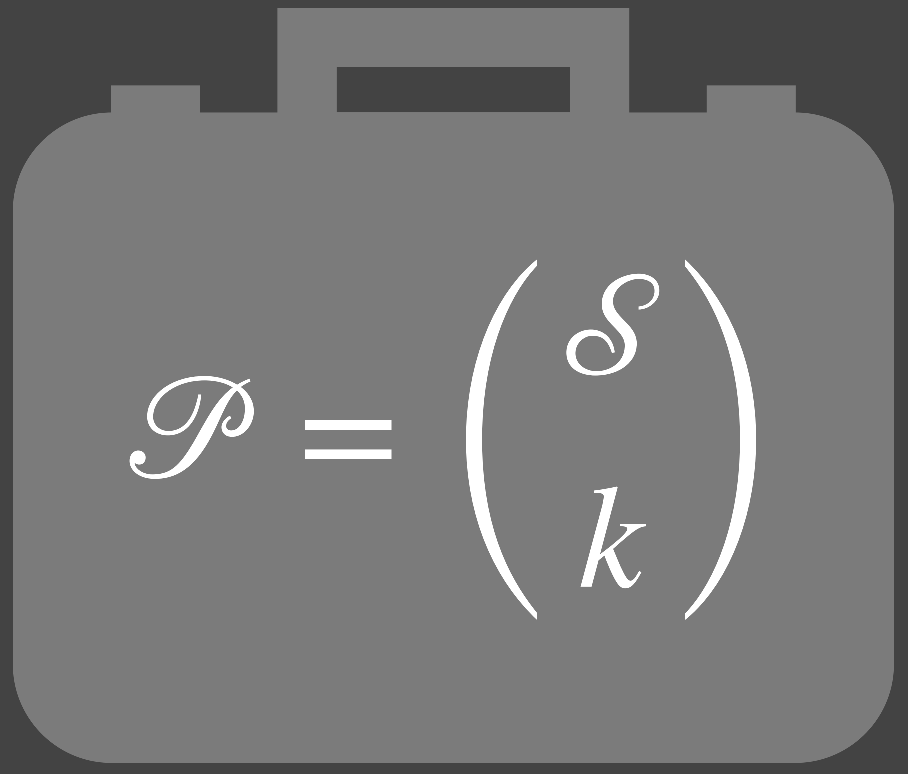
Else:

(I) Buy random set R from $\mathscr{P}$ to cover v.

(II) "Prune" $P \not\ni v$ from $\mathscr{P}$.

LearnOrCover





$k := |OPT|$

@ time t, element v arrives:

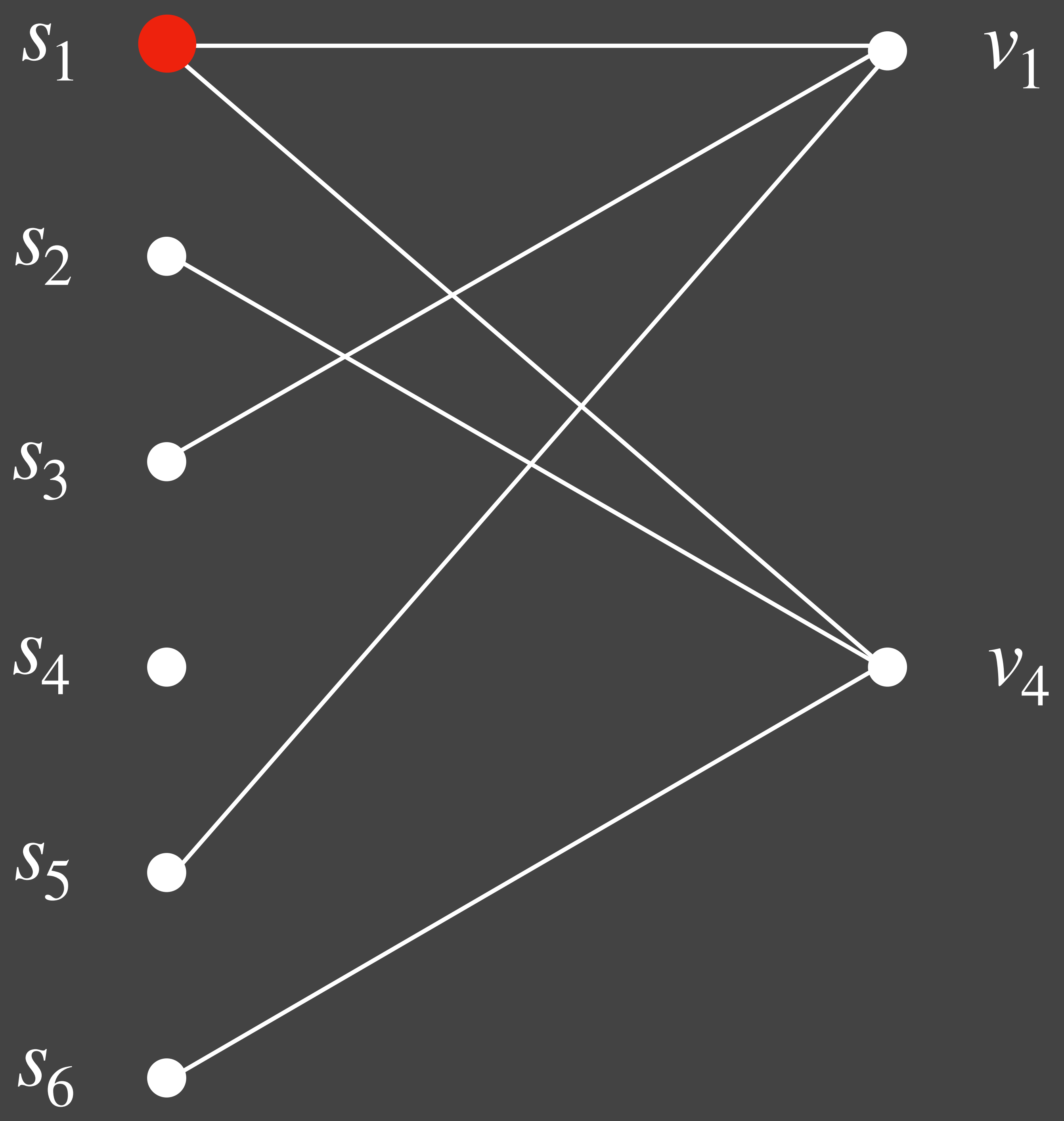
If v covered, do nothing.

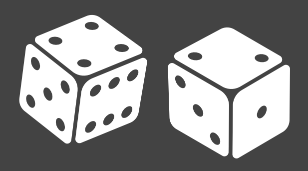
Else:

(I) Buy random set R from $\mathscr{P}$ to cover v.

(II) "Prune" $P \not\ni v$ from $\mathscr{P}$.

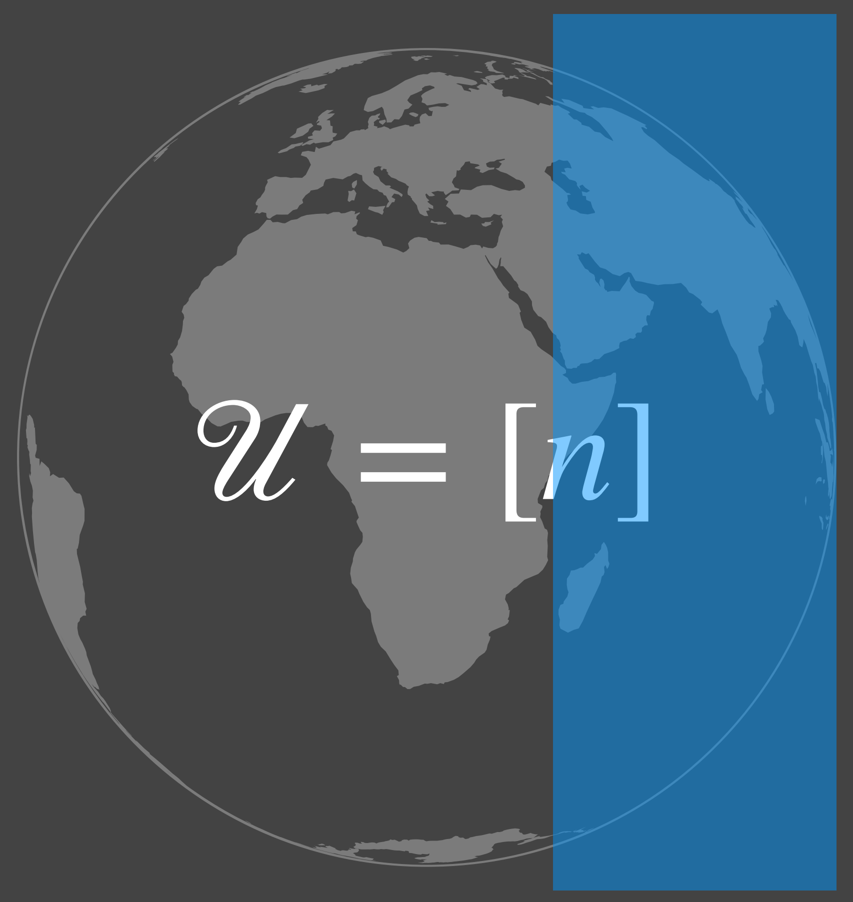
Proof idea: progress learning or covering.

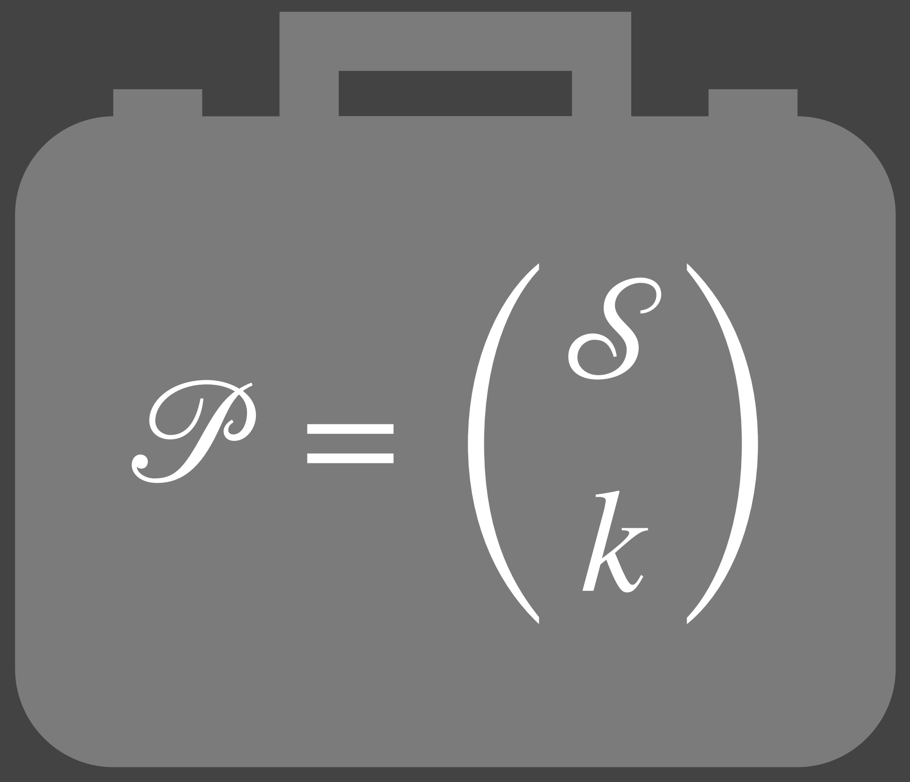




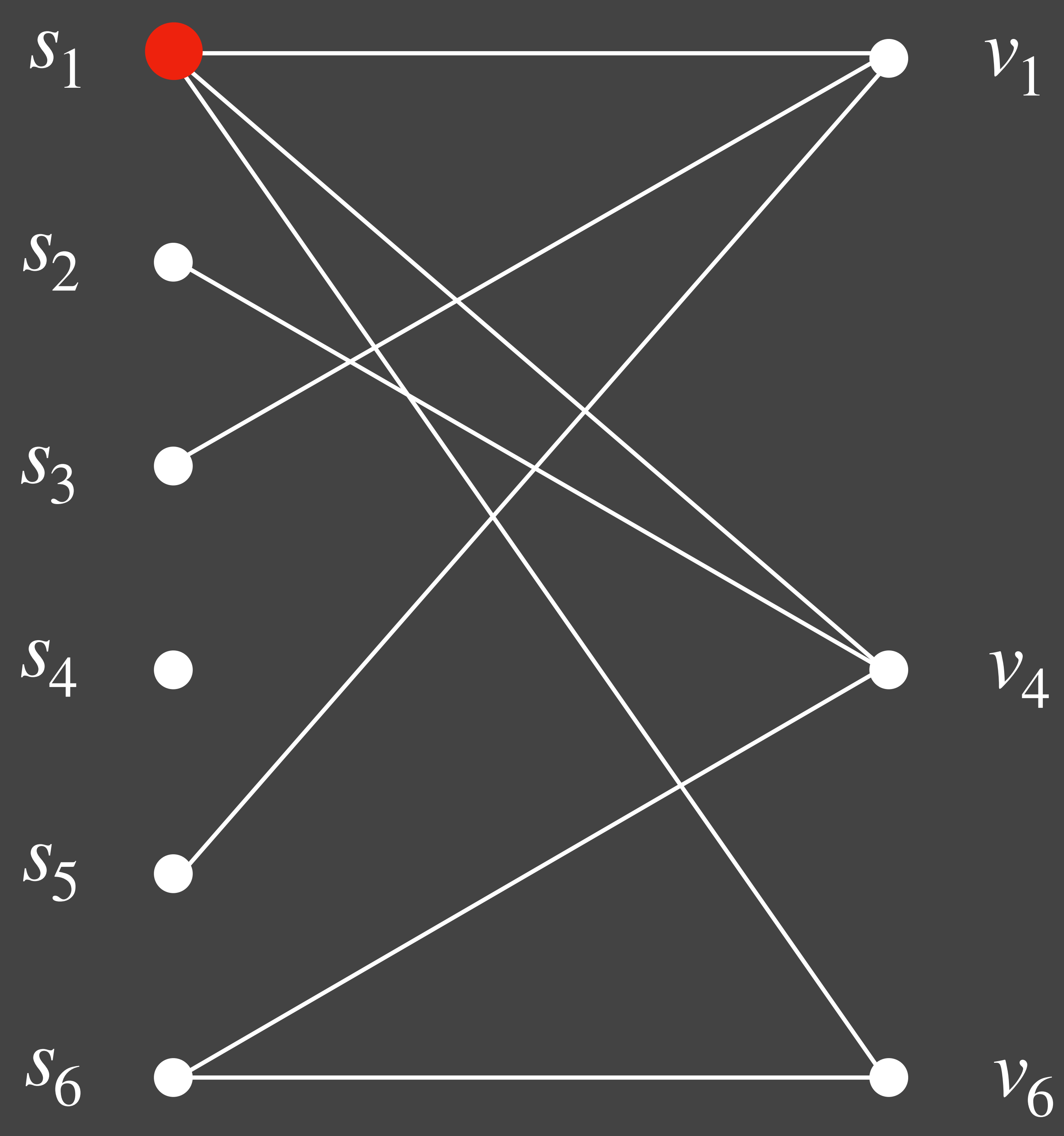
LearnOrCover

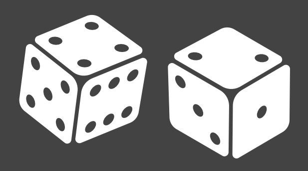
Proof idea: progress learning or covering.





$k := |OPT|$





@ time t, element v arrives:

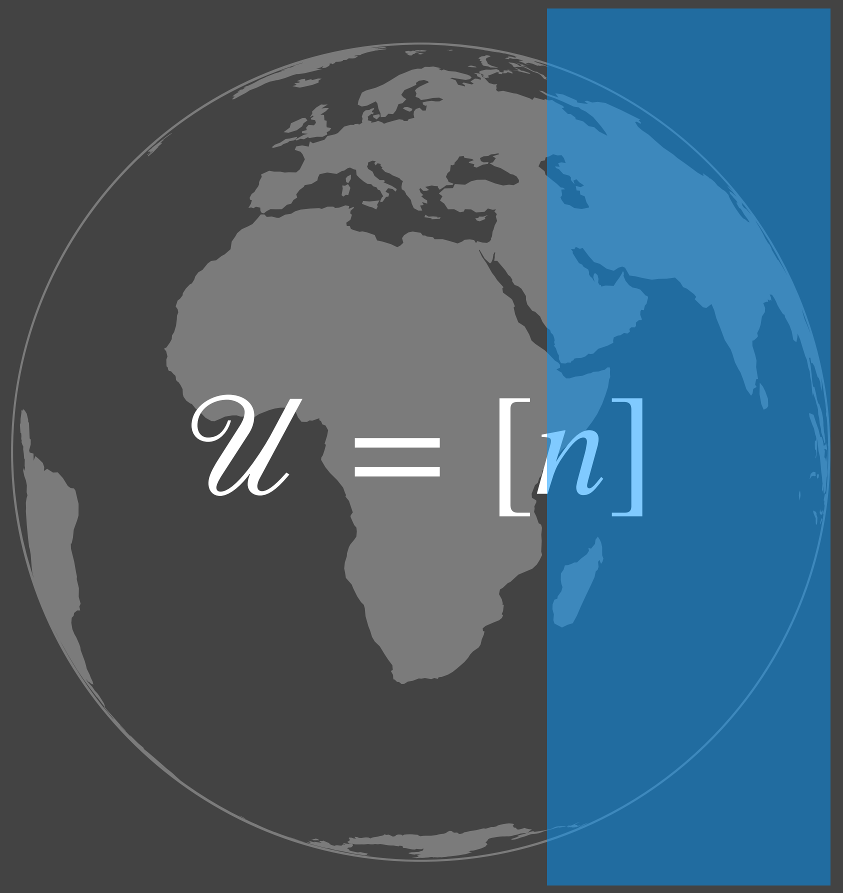
If v covered, do nothing.

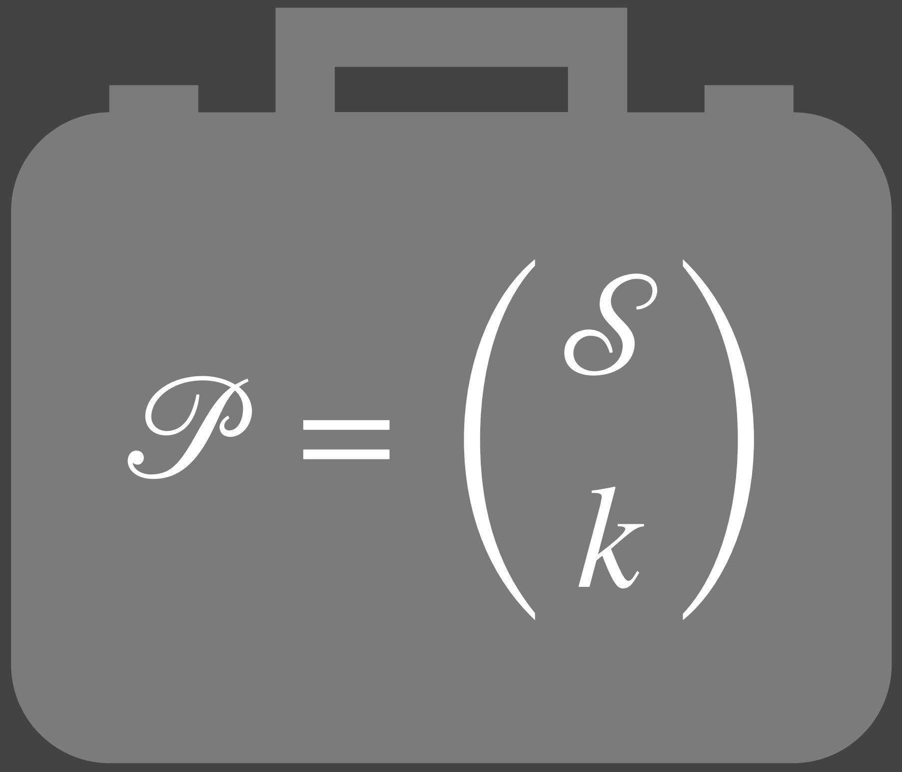
Else:

(I) Buy random set R from $\mathscr{P}$ to cover v.

(II) "Prune" $P \not\ni v$ from $\mathscr{P}$.

LearnOrCover





$k := |OPT|$

@ time t, element v arrives:

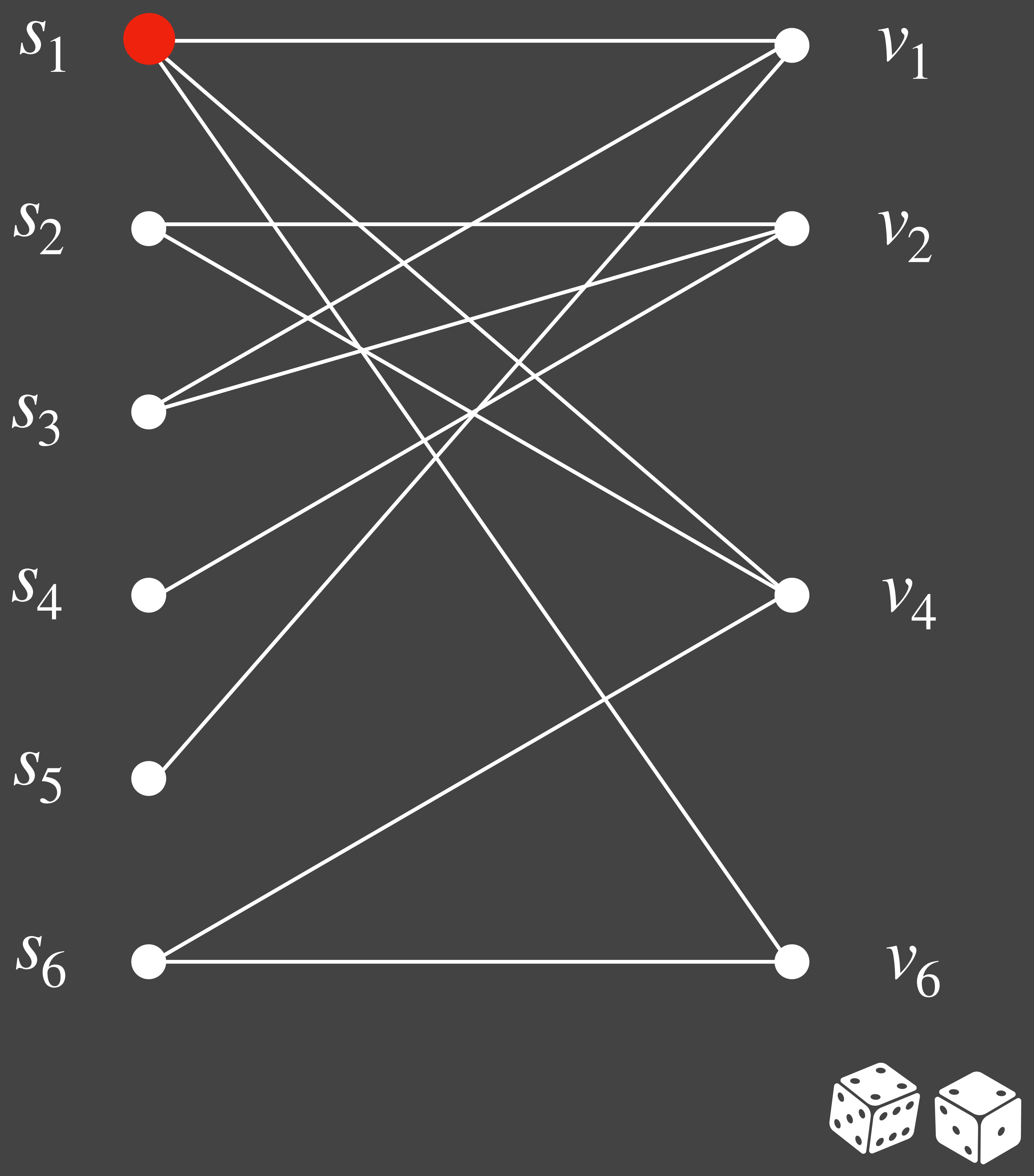
If v covered, do nothing.

Else:

(I) Buy random set R from $\mathscr{P}$ to cover v.

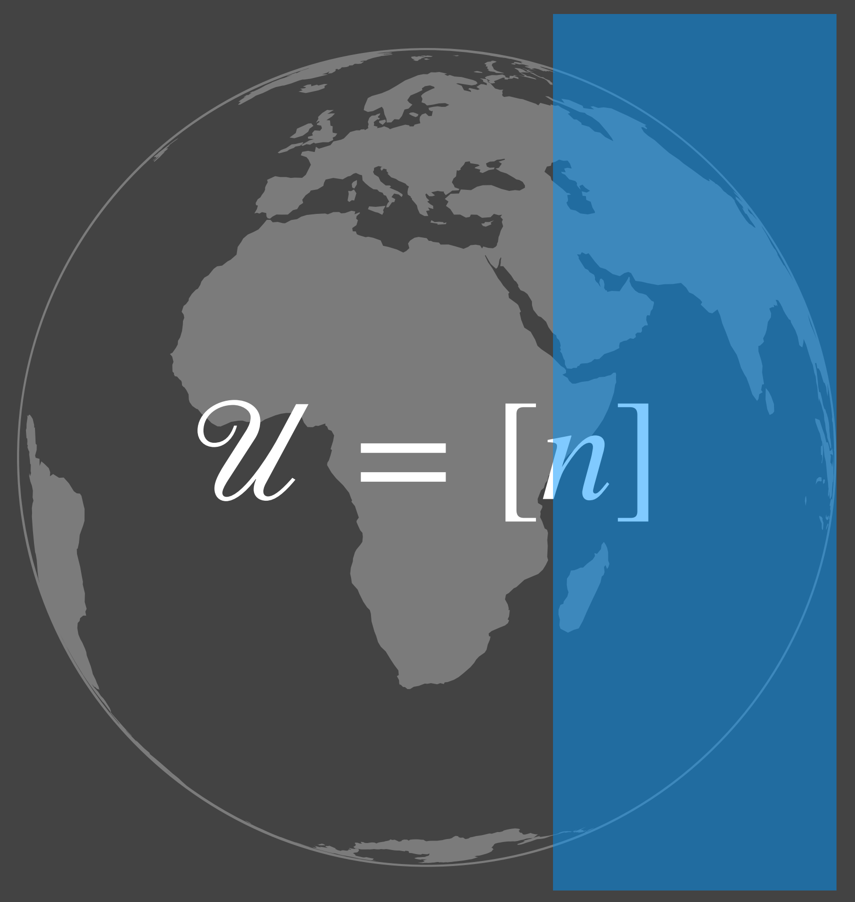
(II) “Prune” $P \not\ni v$ from $\mathscr{P}$.

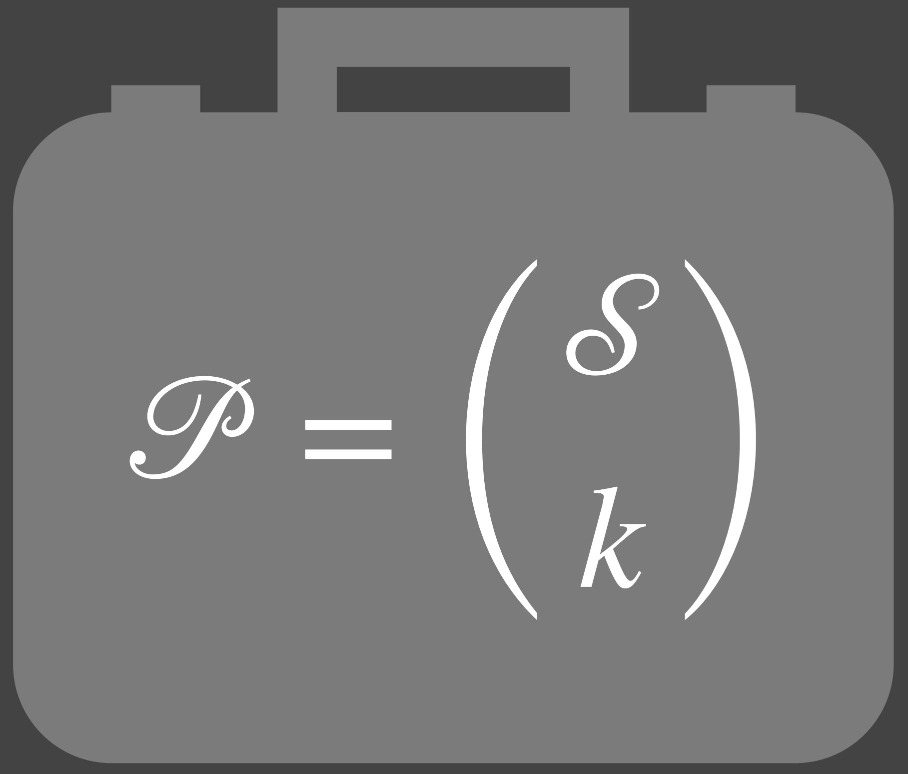
Proof idea: progress learning or covering.



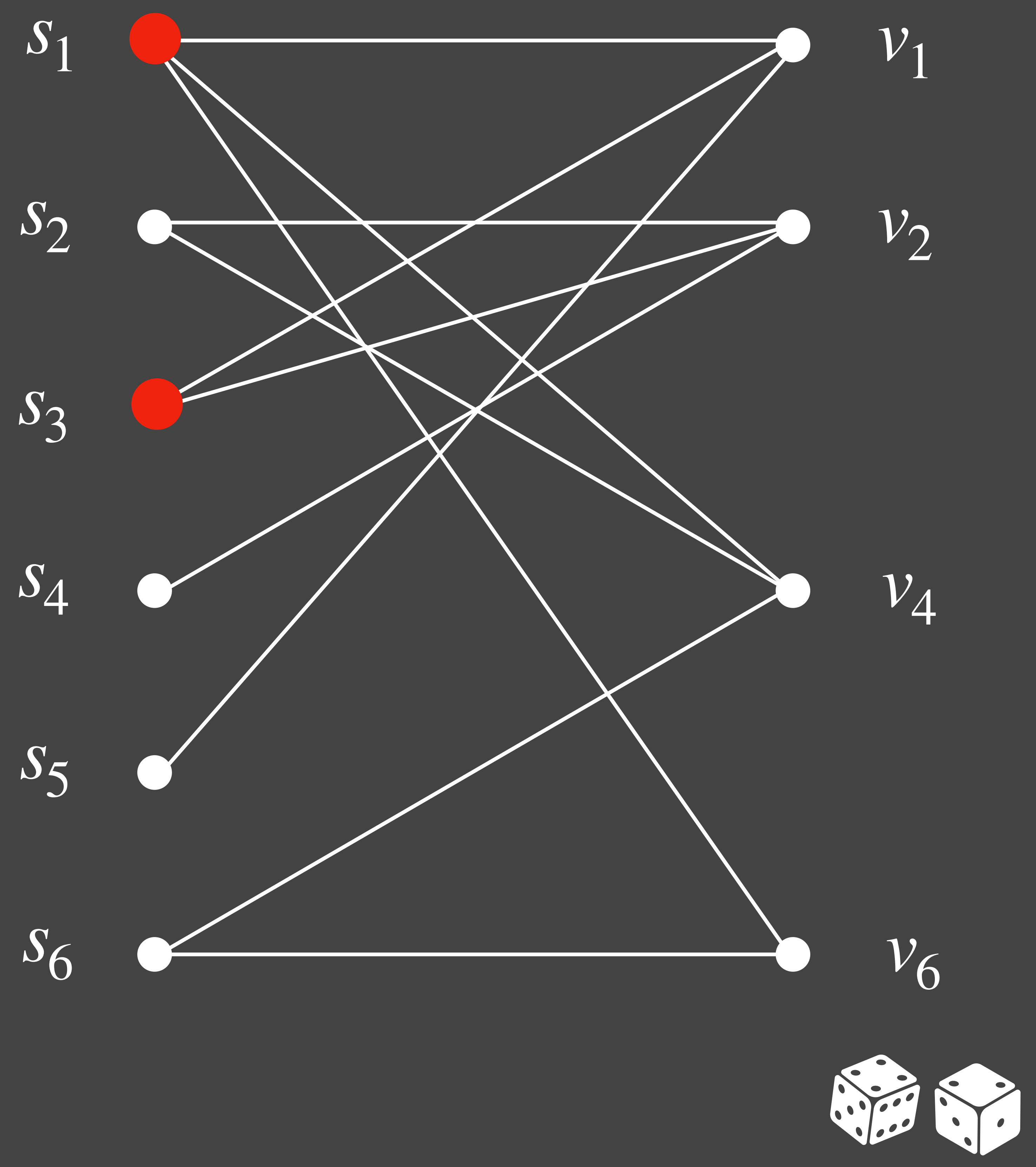
LearnOrCover

Proof idea: progress learning or covering.





$k := |OPT|$



@ time t, element v arrives:

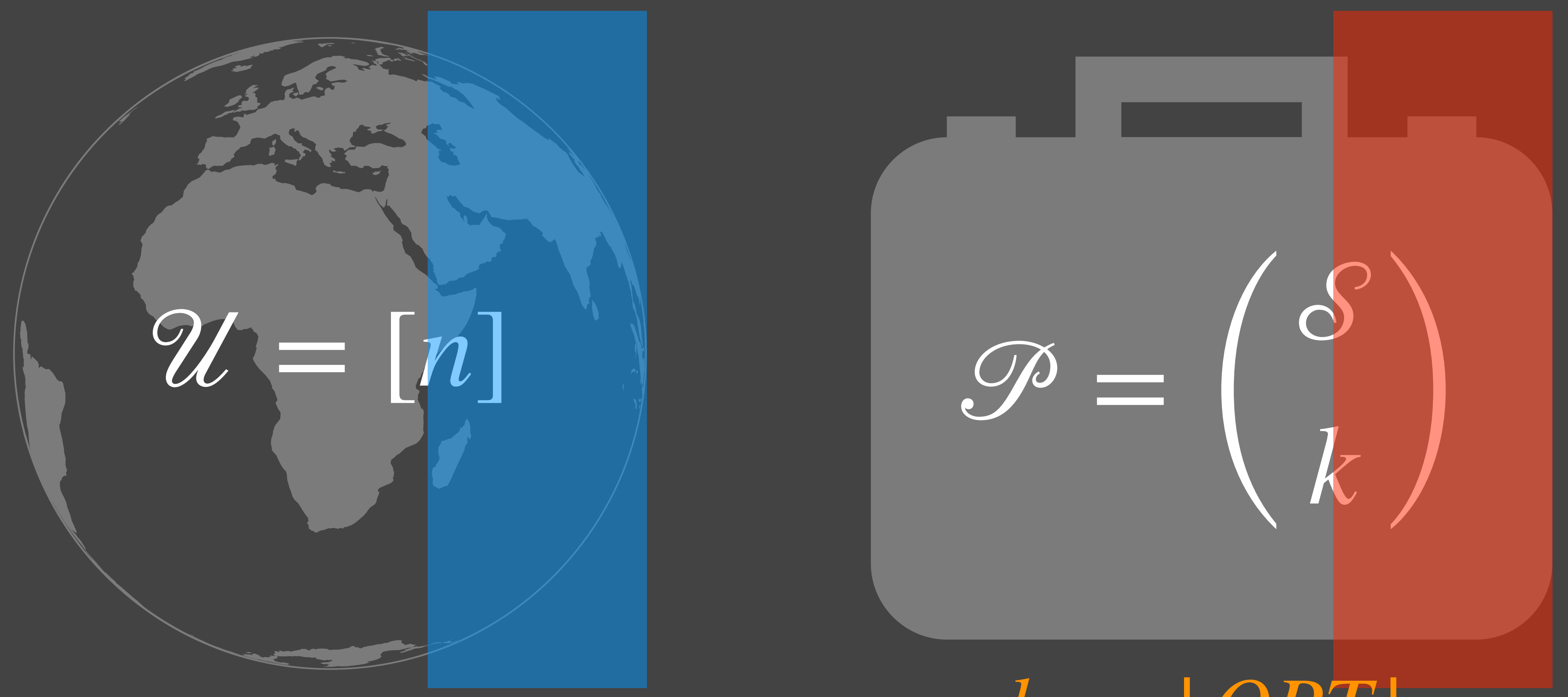
If v covered, do nothing.

Else:

(I) Buy random set R from $\mathscr{P}$ to cover v.

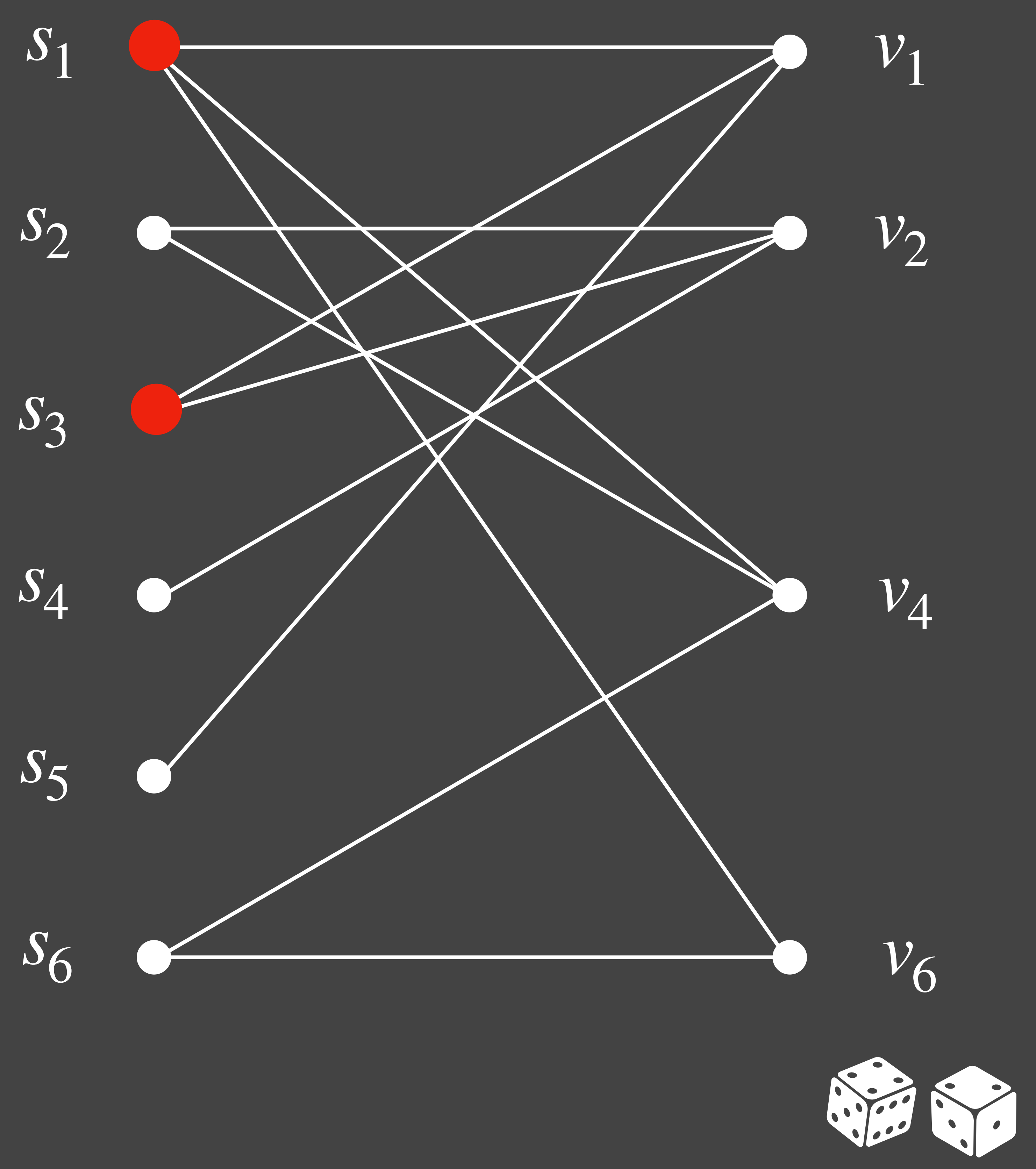
(II) "Prune" $P \not\ni v$ from $\mathscr{P}$.

LearnOrCover



$k := |OPT|$

Proof idea: progress learning or covering.



@ time t, element v arrives:

If v covered, do nothing.

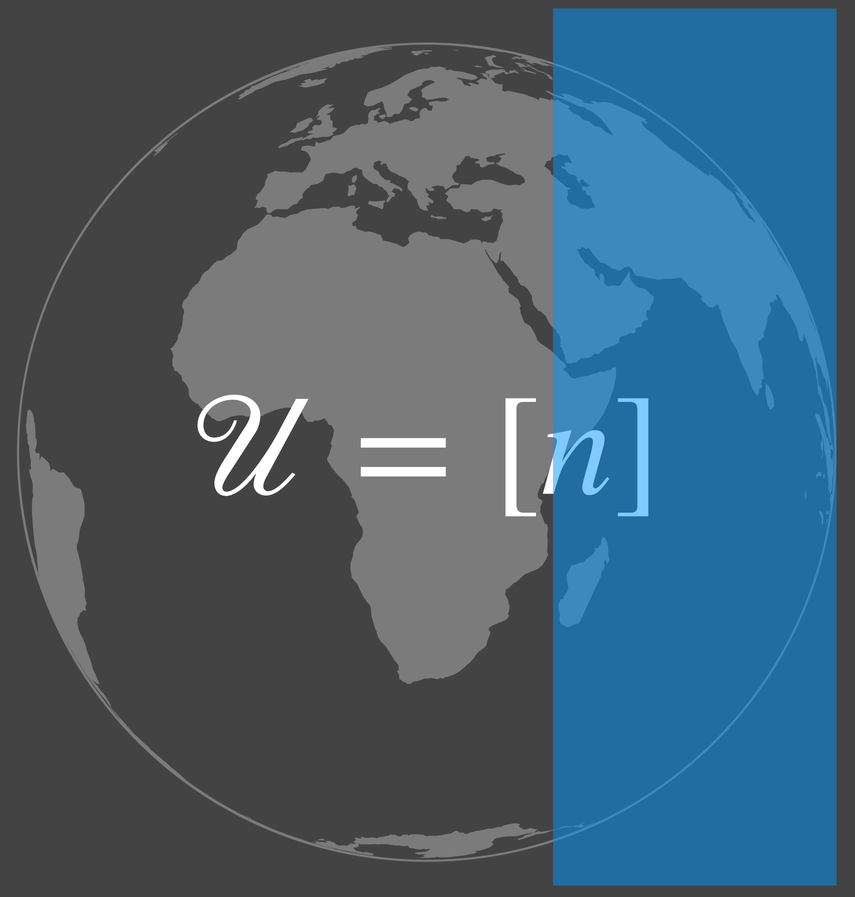
Else:

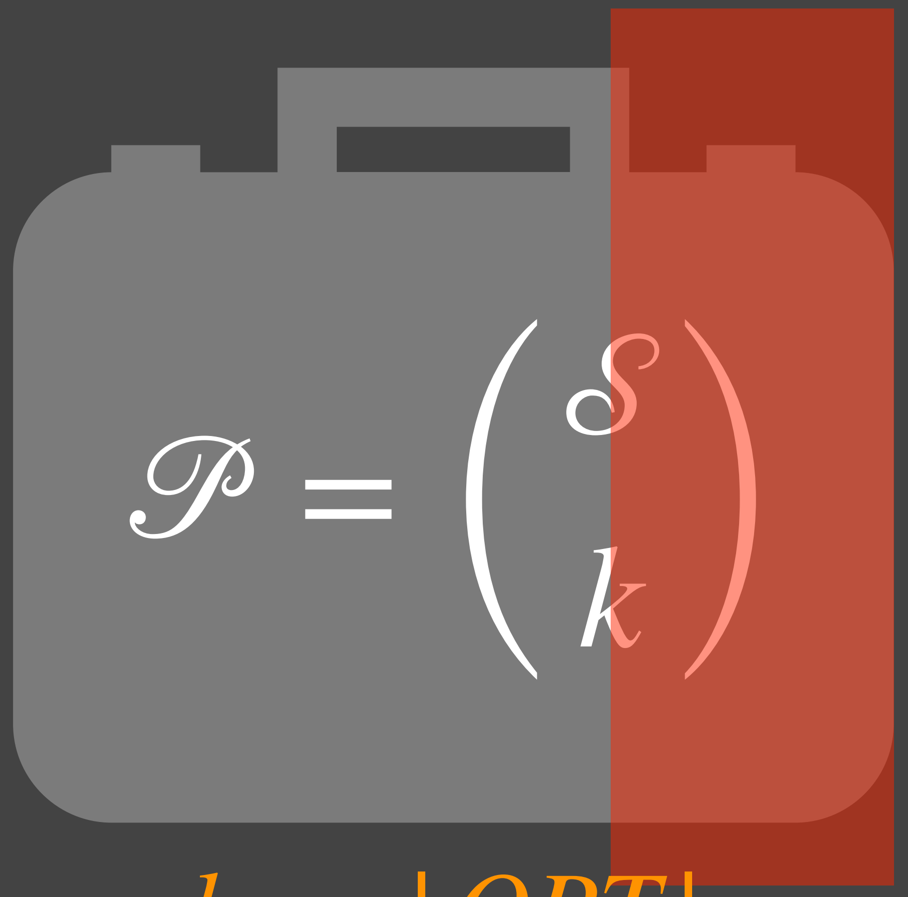
(I) Buy random set R from $\mathscr{P}$ to cover v.

(II) "Prune" $P \not\ni v$ from $\mathscr{P}$.

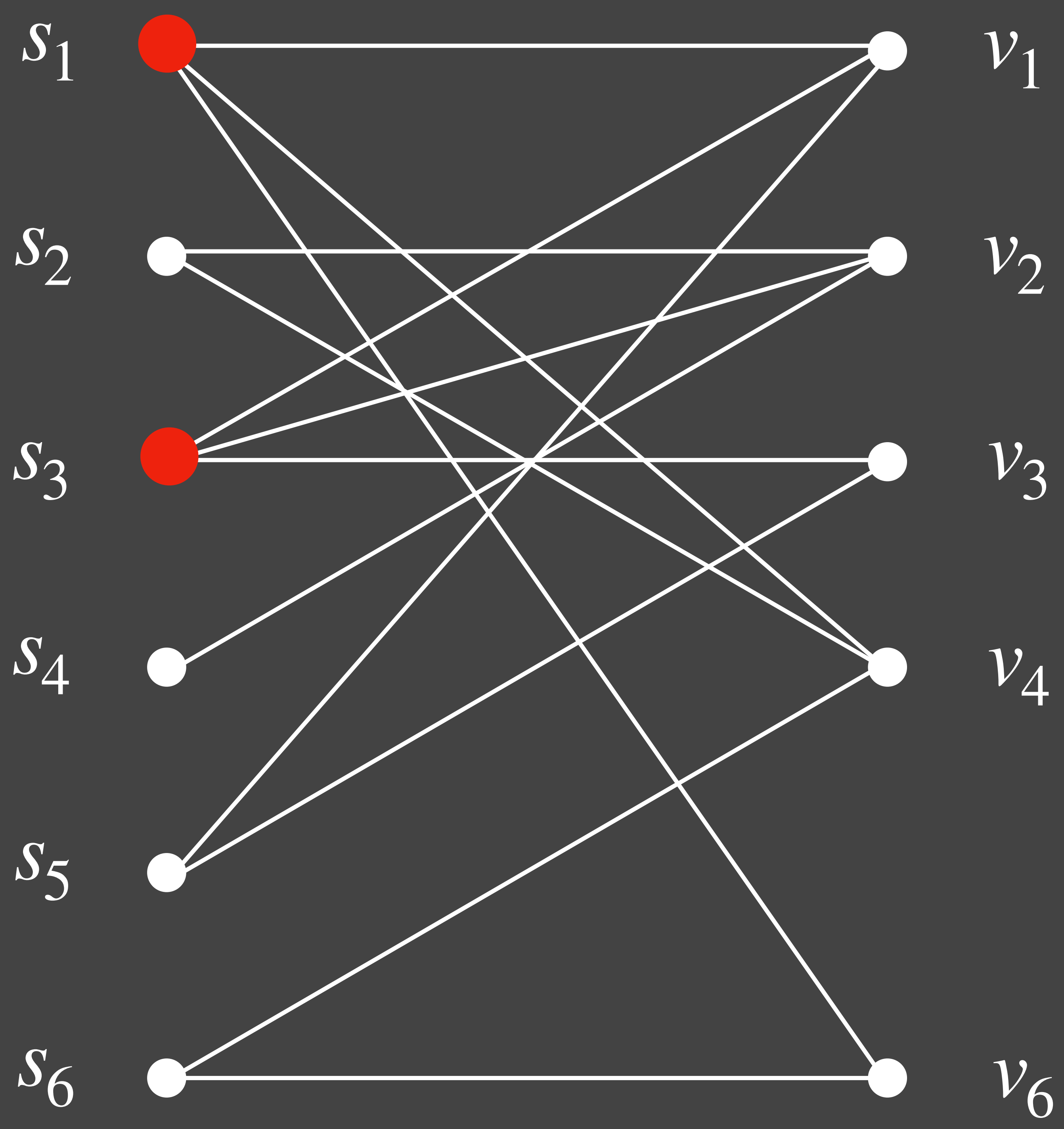
LearnOrCover

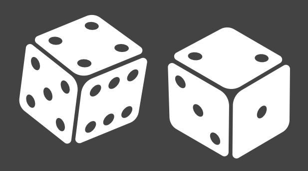
Proof idea: progress learning or covering.





$k := |OPT|$





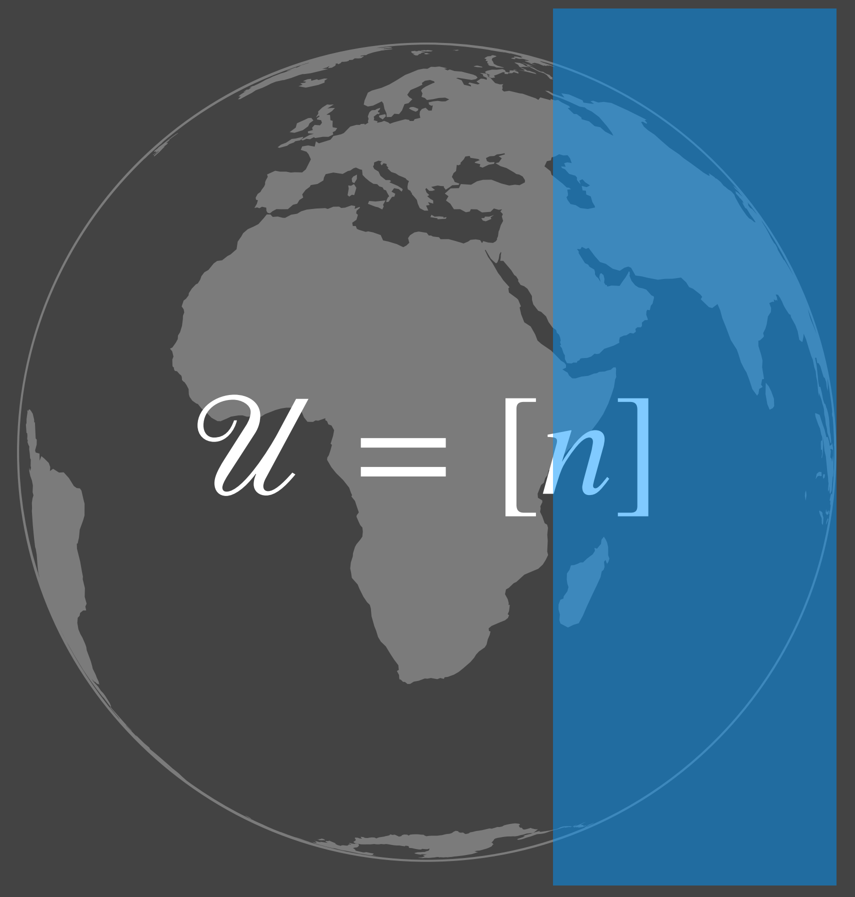
@ time t, element v arrives:

If v covered, do nothing.

Else:

(I) Buy random set R from $\mathscr{P}$ to cover v.
(II) "Prune" $P \not\ni v$ from $\mathscr{P}$.

LearnOrCover



$k := |OPT|$

@ time t, element v arrives:

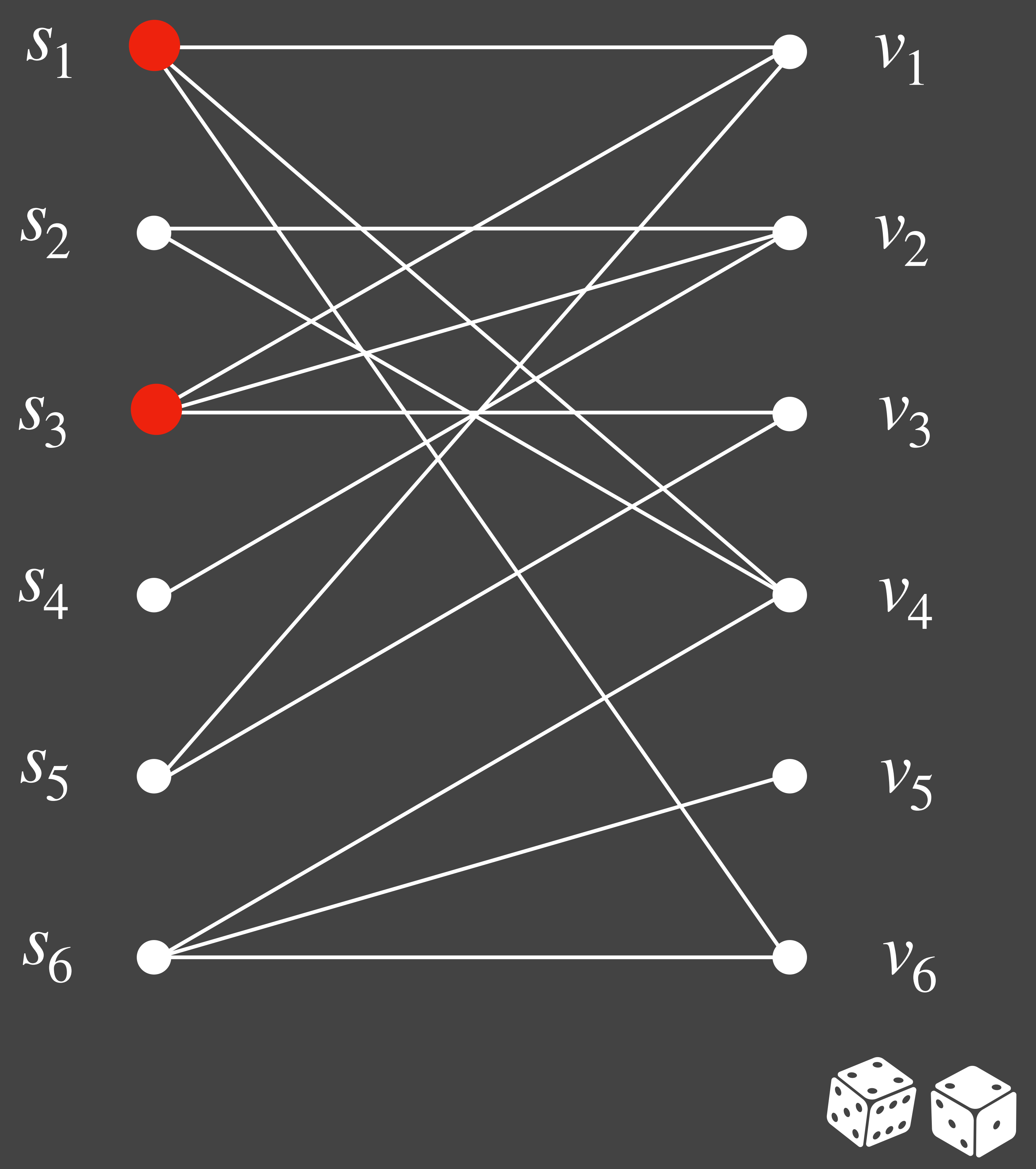
If v covered, do nothing.

Else:

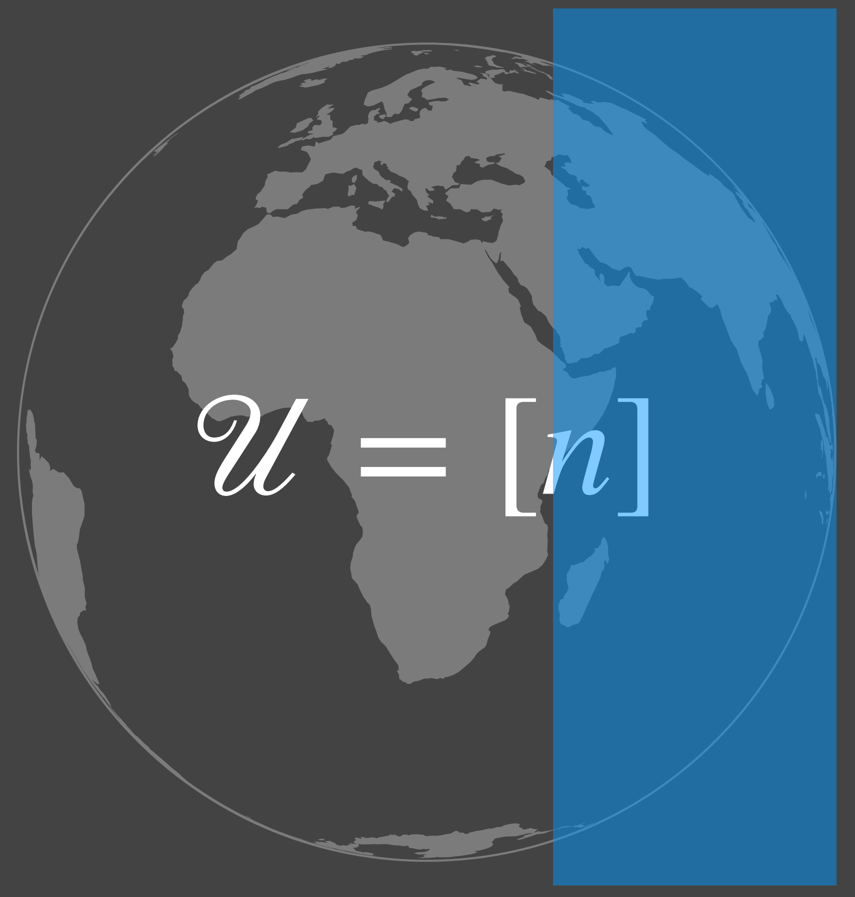
(I) Buy random set R from $\mathscr{P}$ to cover v.

(II) “Prune” $P \not\ni v$ from $\mathscr{P}$.

Proof idea: progress learning or covering.



LearnOrCover



$k := |OPT|$

@ time t, element v arrives:

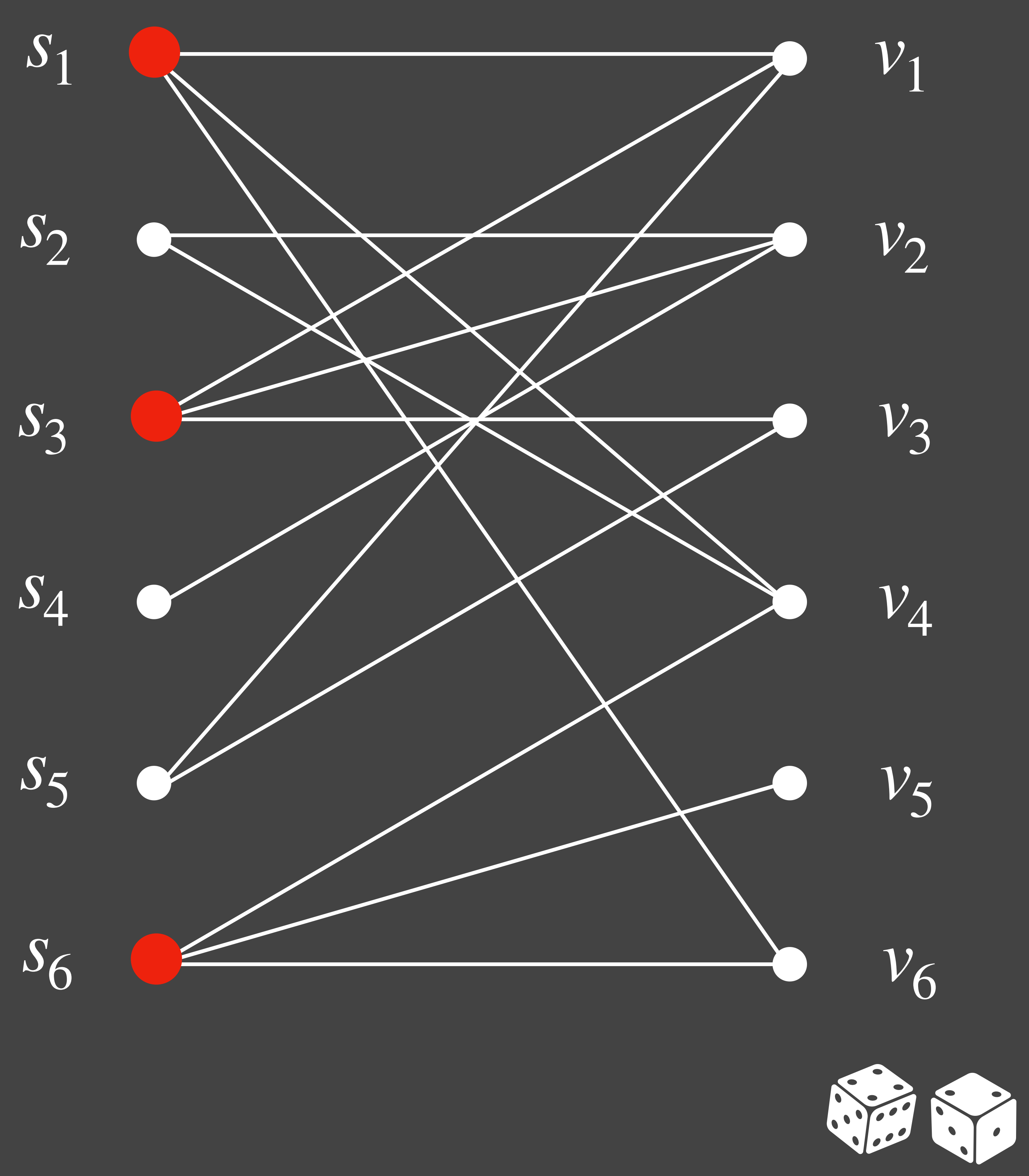
If v covered, do nothing.

Else:

(I) Buy random set R from $\mathscr{P}$ to cover v.

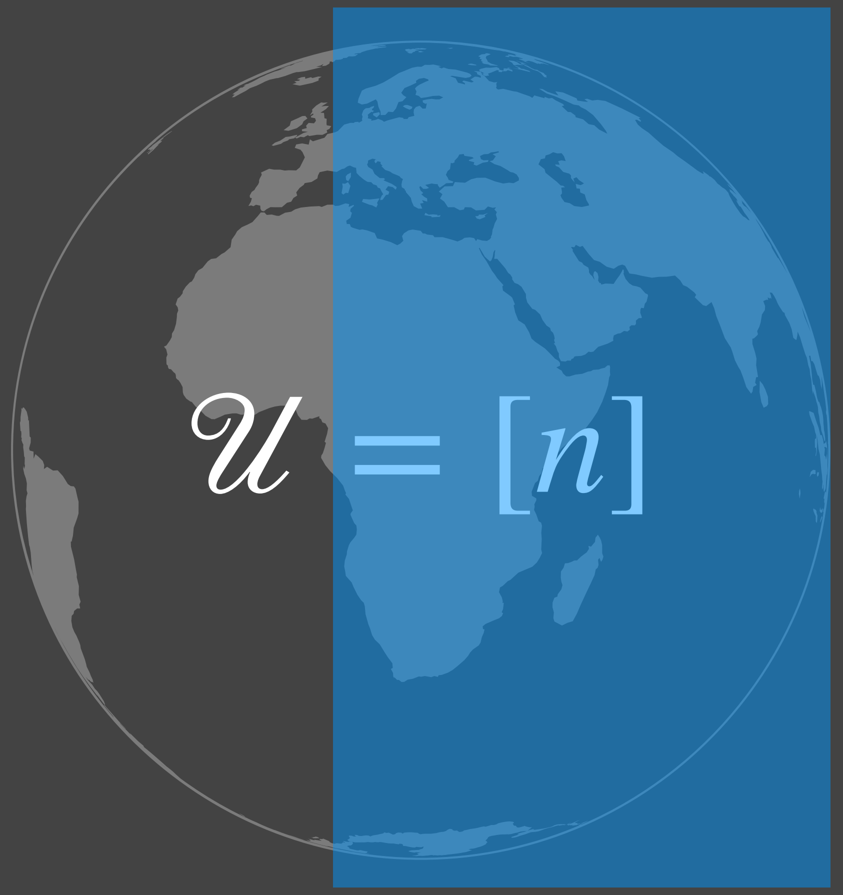
(II) "Prune" $P \not\ni v$ from $\mathscr{P}$.

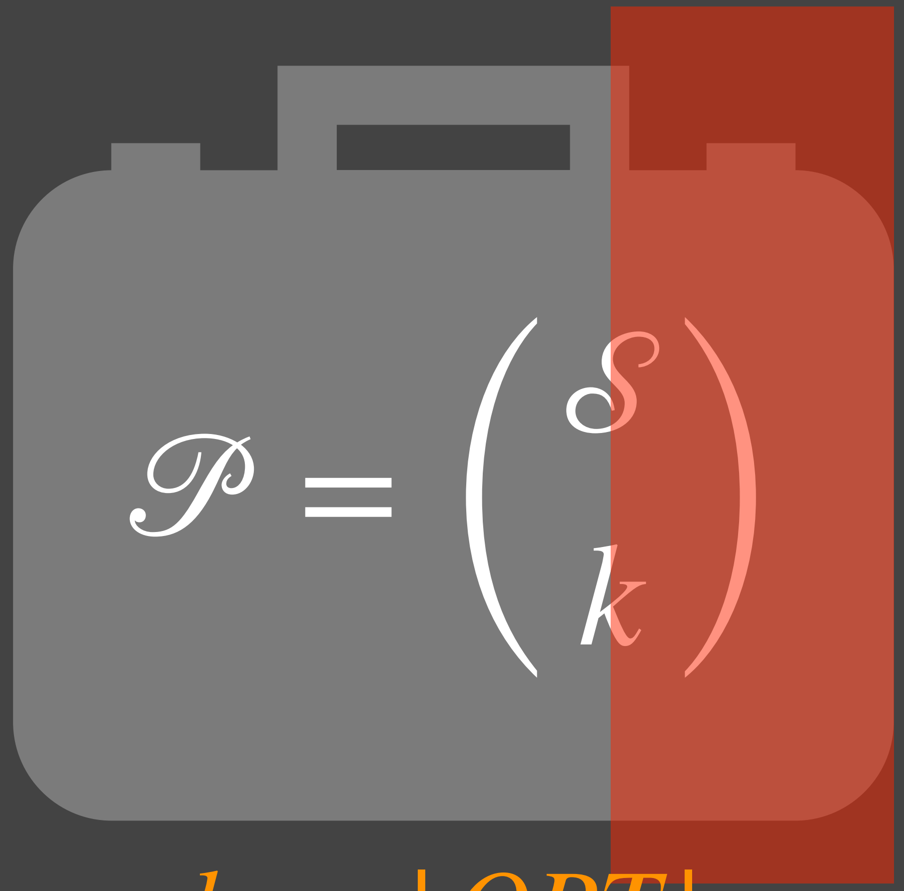
Proof idea: progress learning or covering.



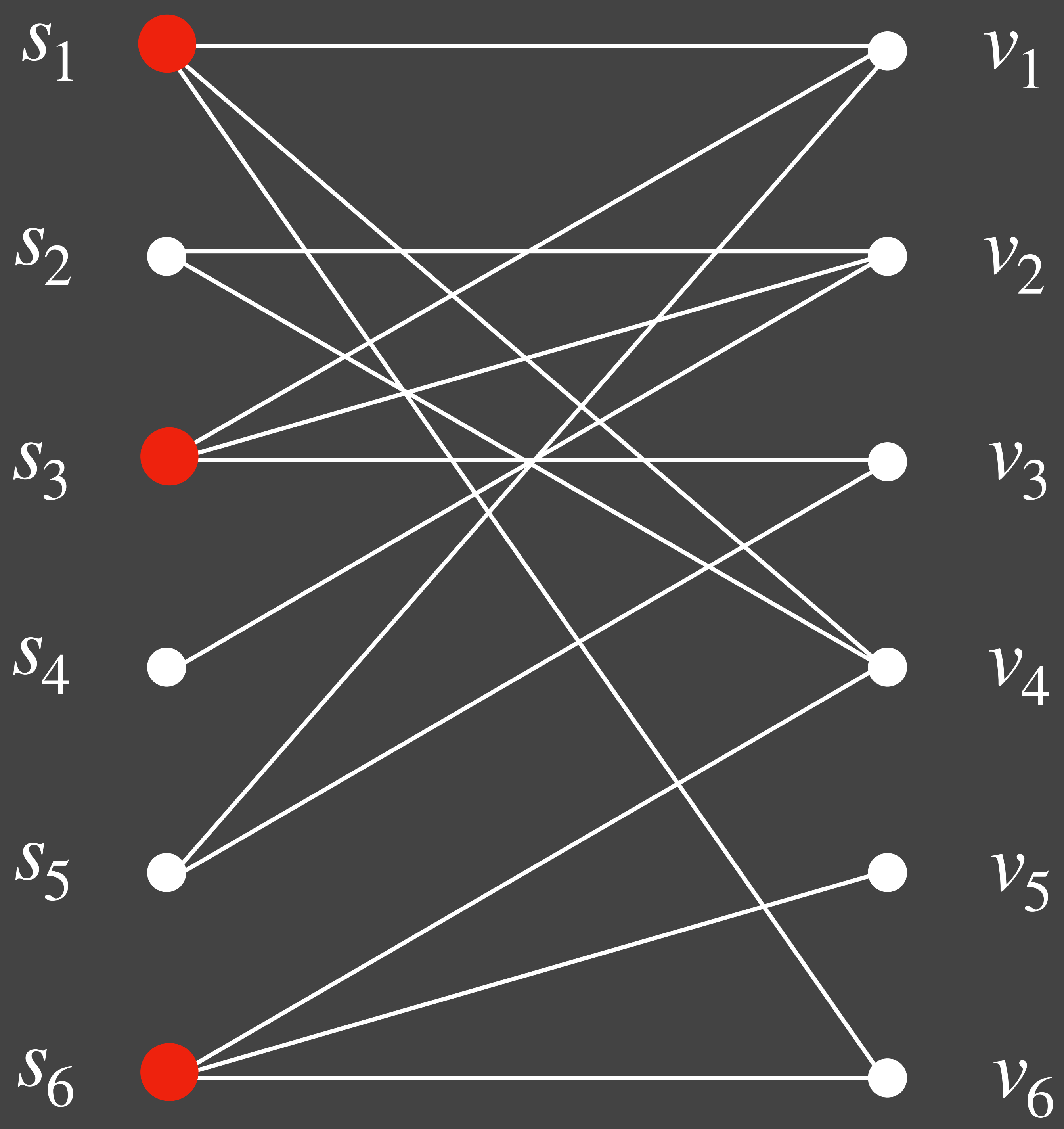
LearnOrCover

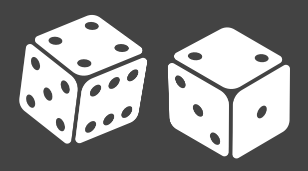
Proof idea: progress learning or covering.





$k := |OPT|$





@ time t, element v arrives:

If v covered, do nothing.

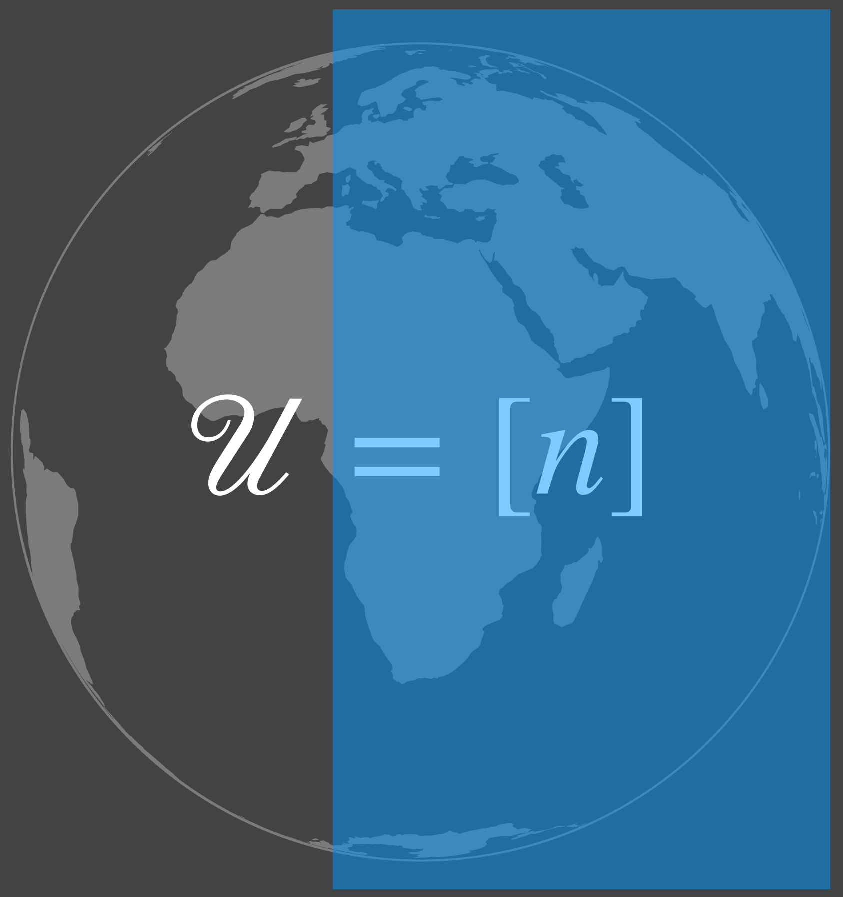
Else:

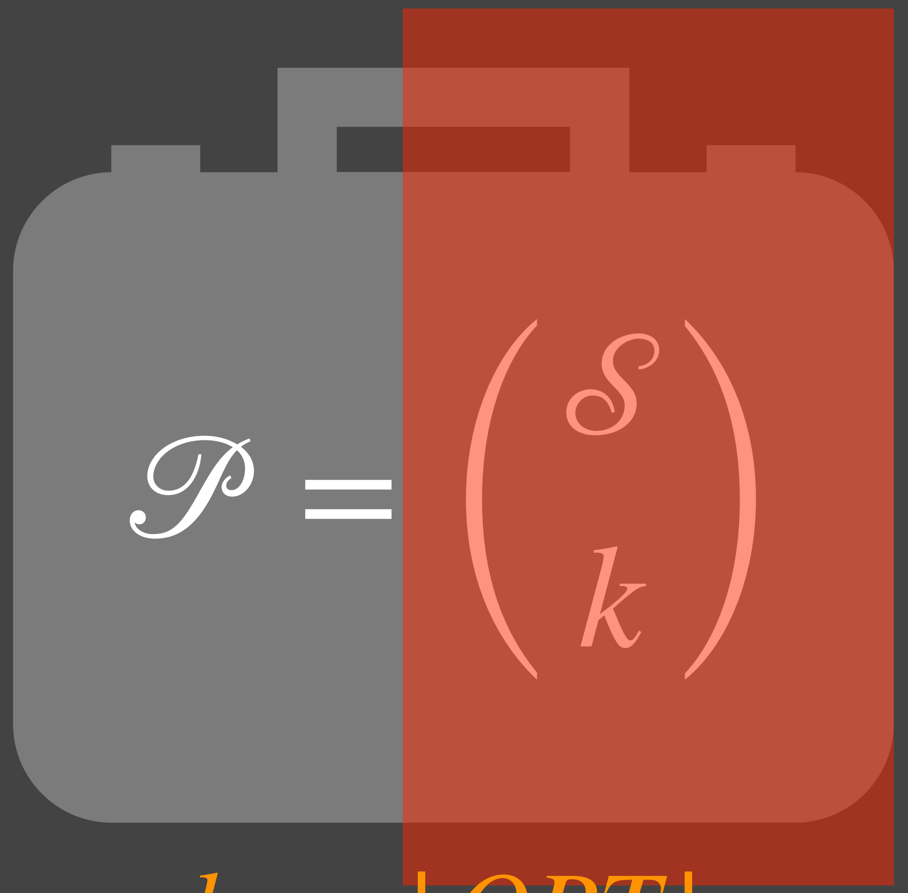
(I) Buy random set R from $\mathscr{P}$ to cover v.

(II) "Prune" $P \not\ni v$ from $\mathscr{P}$.

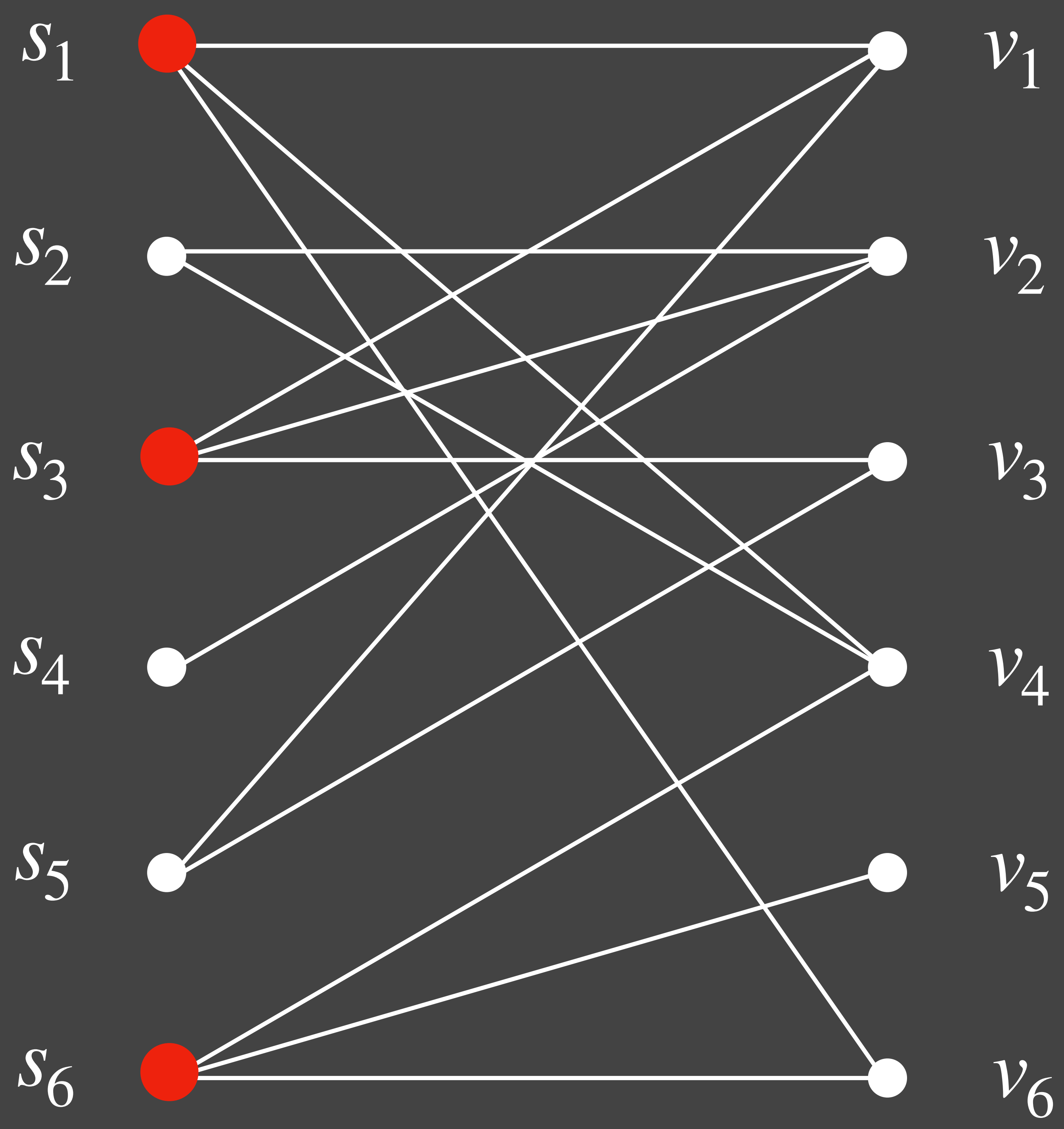
LearnOrCover

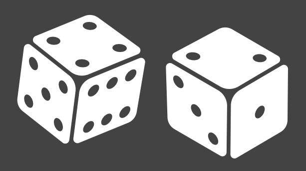
Proof idea: progress learning or covering.





$k := |OPT|$





@ time t, element v arrives:

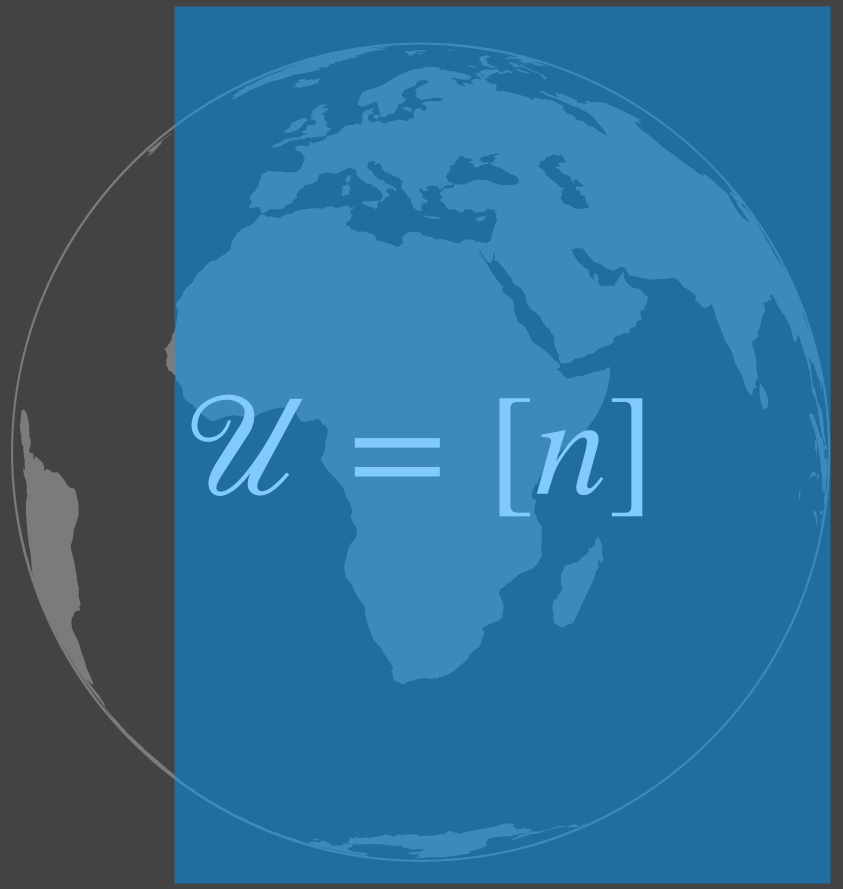
If v covered, do nothing.

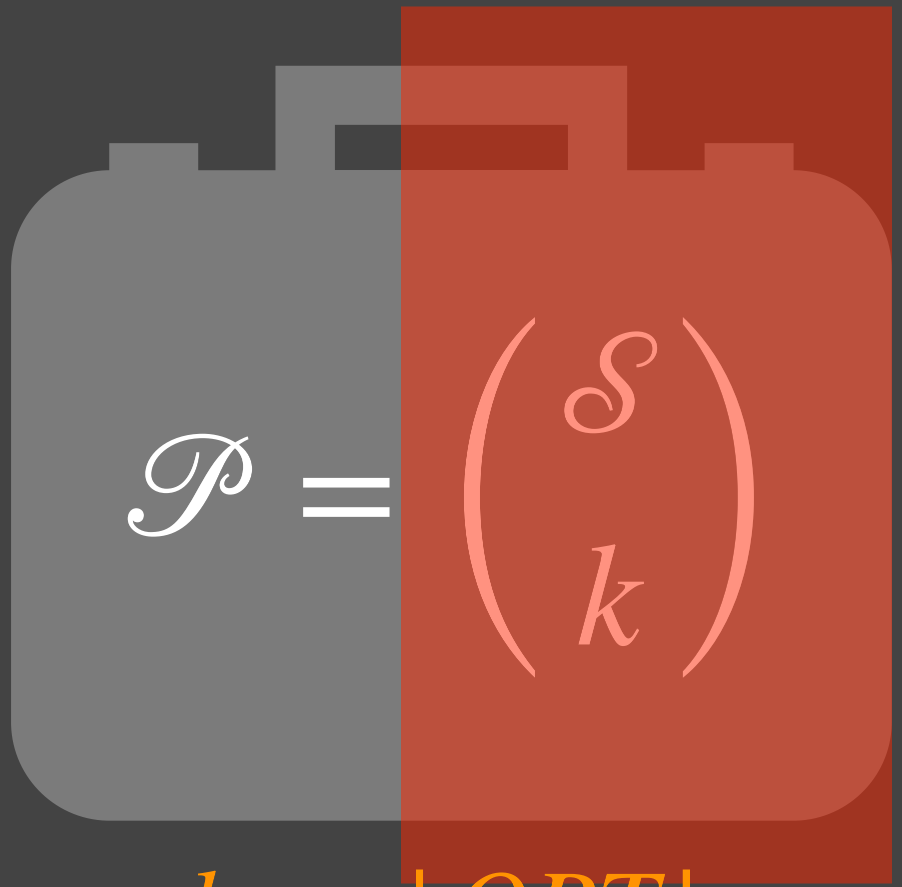
Else:

(I) Buy random set R from $\mathscr{P}$ to cover v.

(II) “Prune” $P \not\ni v$ from $\mathscr{P}$.

LearnOrCover





$k := |OPT|$

@ time t, element v arrives:

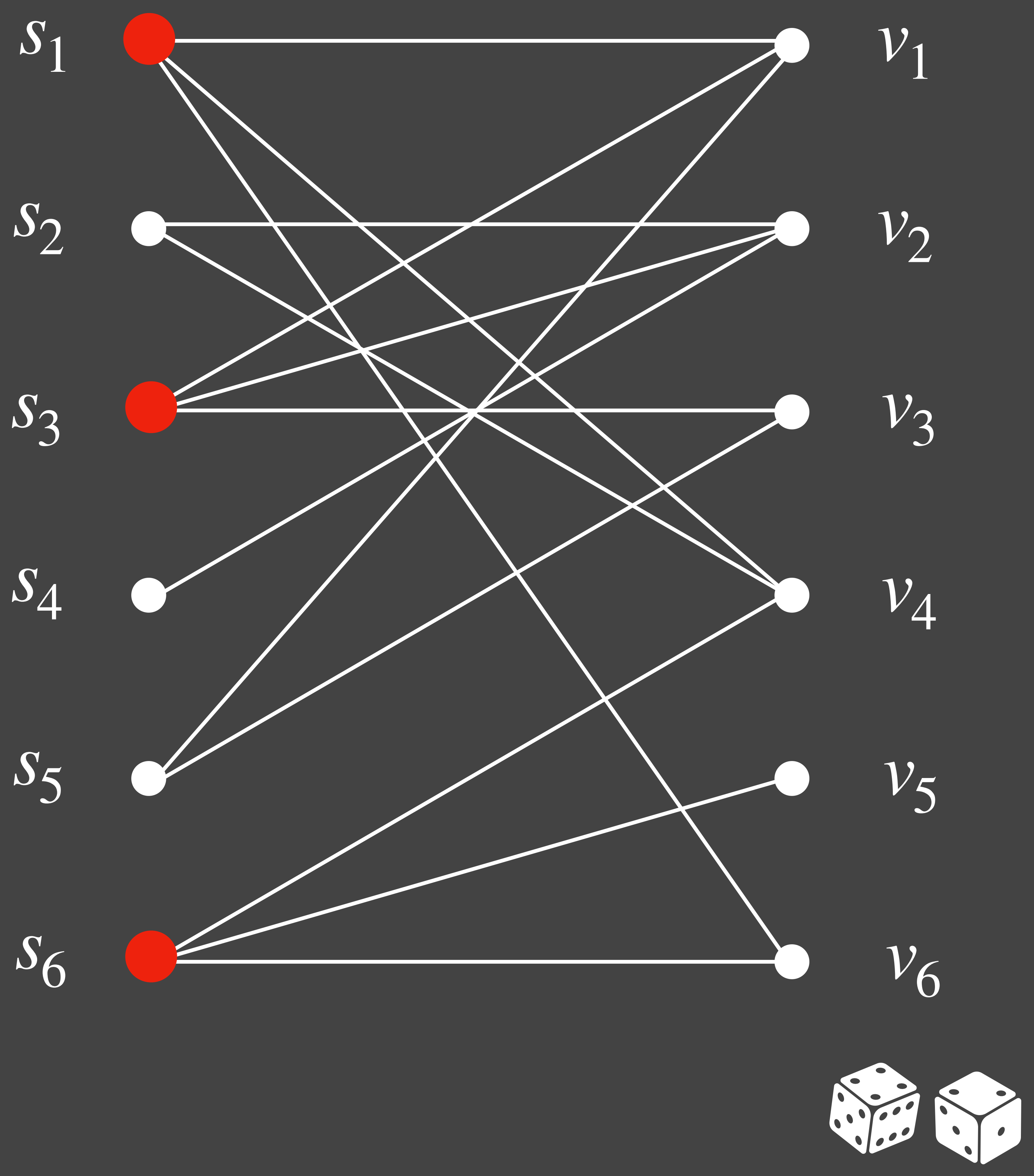
If v covered, do nothing.

Else:

(I) Buy random set R from $\mathscr{P}$ to cover v.

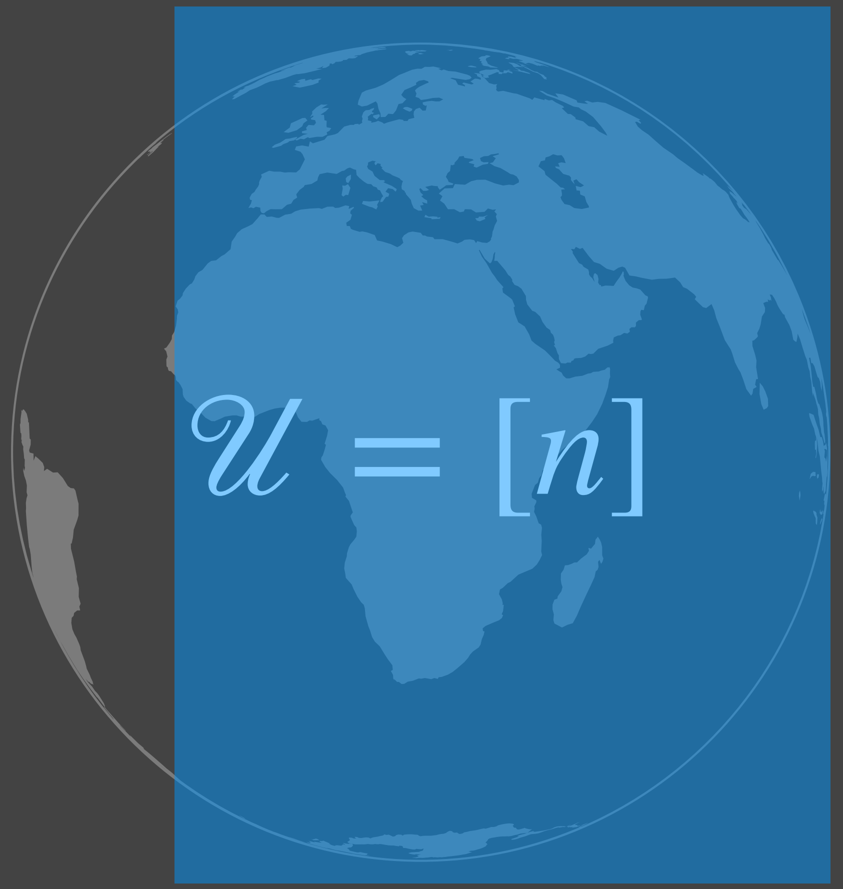
(II) "Prune" $P \not\ni v$ from $\mathscr{P}$.

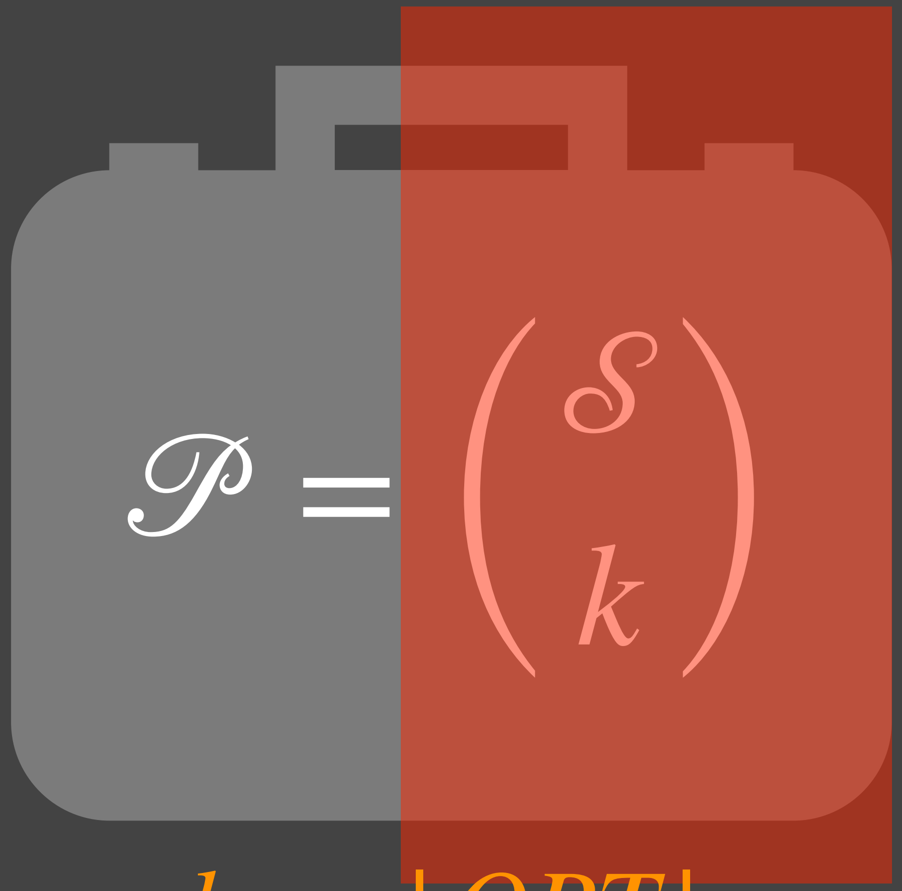
Proof idea: progress learning or covering.



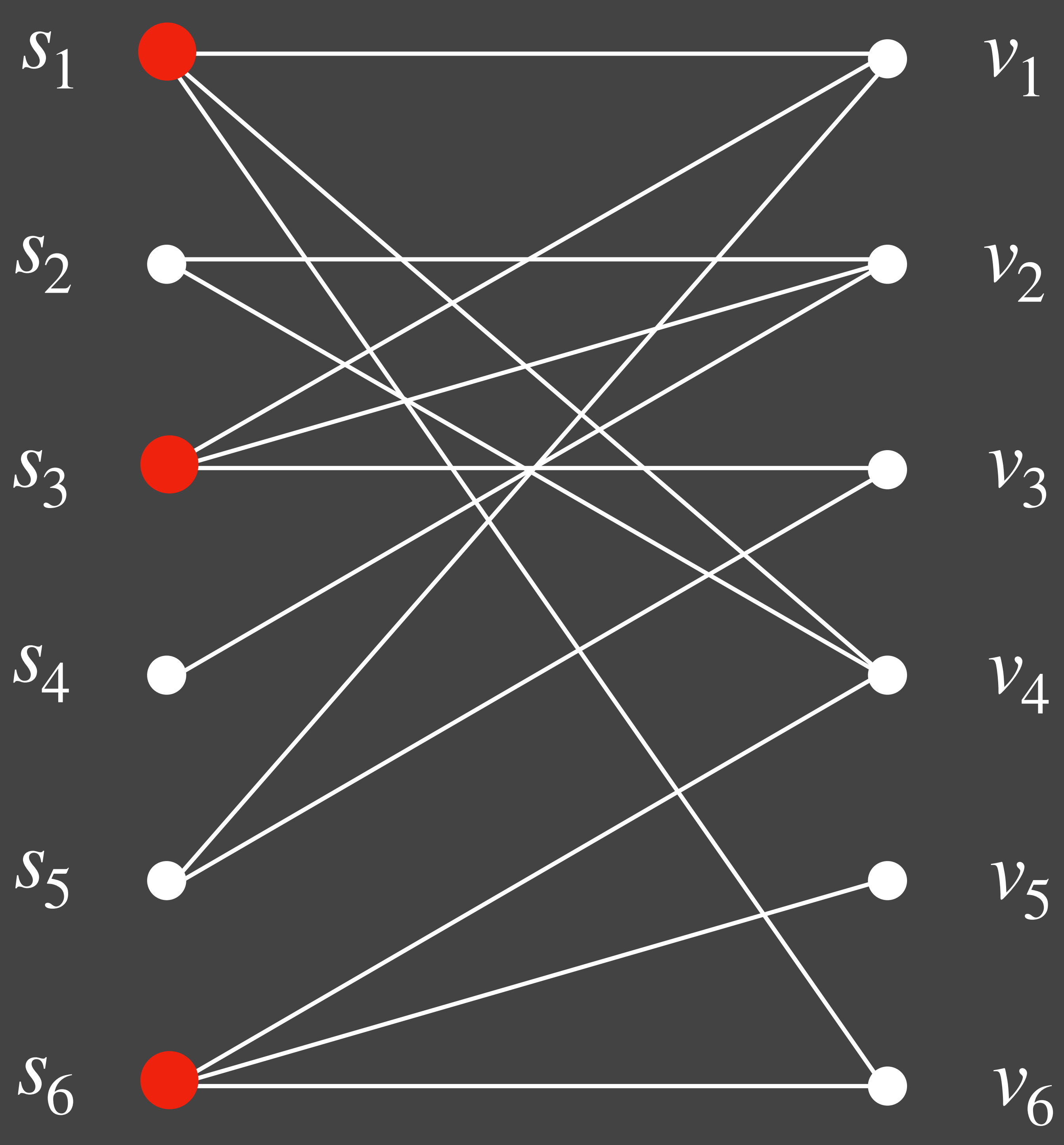
LearnOrCover

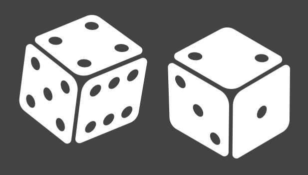
Proof idea: progress learning or covering.





$k := |OPT|$





@ time t, element v arrives:

If v covered, do nothing.

Else:

(I) Buy random set R from $\mathcal{P}$ to cover v.
(II) "Prune" $P \not\ni v$ from $\mathcal{P}$.

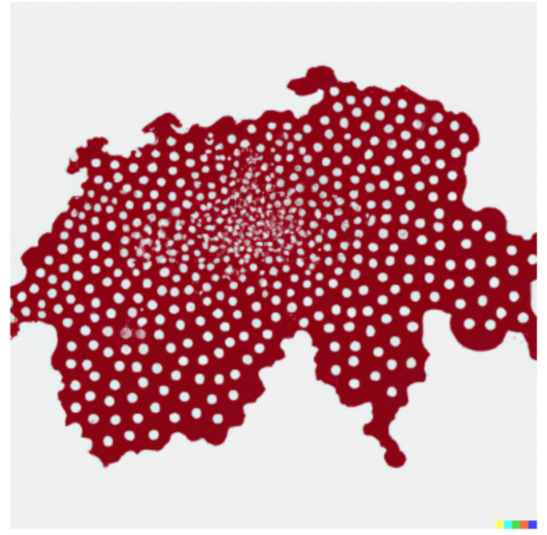
After $O(\log n) \cdot$ OPT steps,

$|\mathcal{U}| = 0$ or $|\mathcal{P}| = 1$.

LearnOrCover [GKL. 21] enters the canon

In syllabus of Algorithmic Foundations course @ EPFL

Algorithmic Toolbox --- How to Solve Set Cover in x Ways



Credits	2
Lecturer	Ola Svensson
Office hours	Wednesdays 14:00 - 16:00 in INJ 112
Schedule	Mondays 14-16 in INM201.

Short description

The goal of this PhD course is to give PhD students a toolbox of algorithmic techniques in order to successfully address their favorite problems. The course emphases the illustration of the main ideas of these techniques. We prefer simplicity over details and we illustrate the algorithmic techniques in the simple and clean setting of the set cover problem. The algorithmic techniques that we plan to cover include

- Greedy algorithms
- Local search algorithms
- Linear programming
  - Randomized rounding (independent, threshold, exponential clocks)
  - Duality (primal-dual algorithms, dual fitting, and the use of complementarity slackness)
- Multiplicative weight update
- Online algorithms in adversarial and random order streams (primal-dual, potential function, and projection based)

In addition, to attending the lectures, students are required to submit a project report where they apply one of the algorithmic techniques in a more complex setting.

Schedule and references

- **Lecture 1 (Monday February 27):** *Introduction. Greedy and Local Search Algorithms. References: Greedy algorithm, Local Search Algorithm (Section 2.1)*
- **Lecture 2 (Monday March 6):** *Linear programming, Threshold and Randomized rounding. References: LPs and Threshold Rounding, Independent Randomized Rounding, see also for a very nice analysis.*
- **Lecture 3 (Monday March 13):** *Exponential clocks, TU matrices, VC-dimension. References: Appendix A for exponential clocks, TU matrices and consecutive ones property , for VC-dimension see here and here*
- **Lecture 4 (Monday March 20):** *TU matrices, VC-dimension. References: Ola's notes*

Take Away III

[Gupta Kehne L. FOCS 21]
[Gupta Kehne L. In Submission]

Q: What happens beyond the worst case?

Take Away III

[Gupta Kehne L. FOCS 21]
[Gupta Kehne L. In Submission]

Q: What happens beyond the worst case?

A1: Random order is as easy as offline.

Take Away III

[Gupta Kehne L. FOCS 21]
[Gupta Kehne L. In Submission]

Q: What happens beyond the worst case?

A1: Random order is as easy as offline.

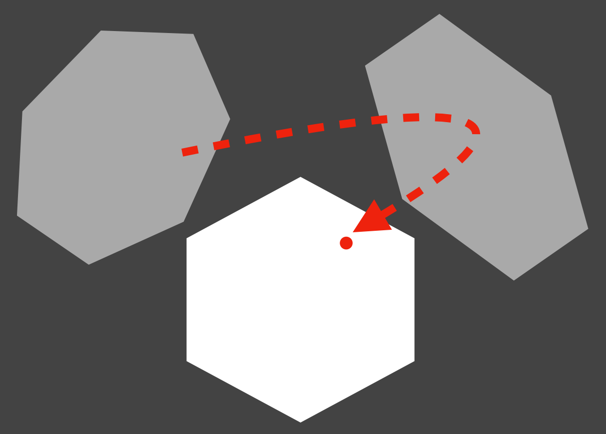
A2: Random instance is as easy as offline.

Outline

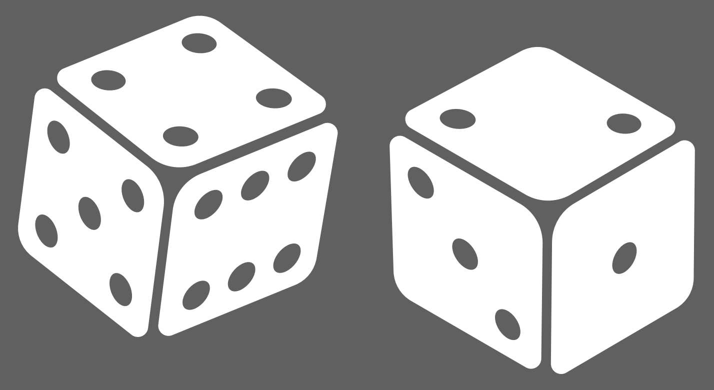
Theme I — Submodular Optimization

f(🍕 | 🥕) ≥ f(🍕 | 🥕, 🍩)

Theme II — Stable Algorithms



Theme III — Beyond Worst-Case Analysis



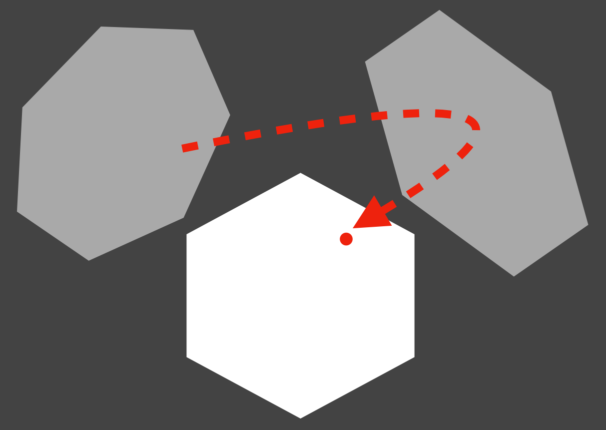
Conclusion

Outline

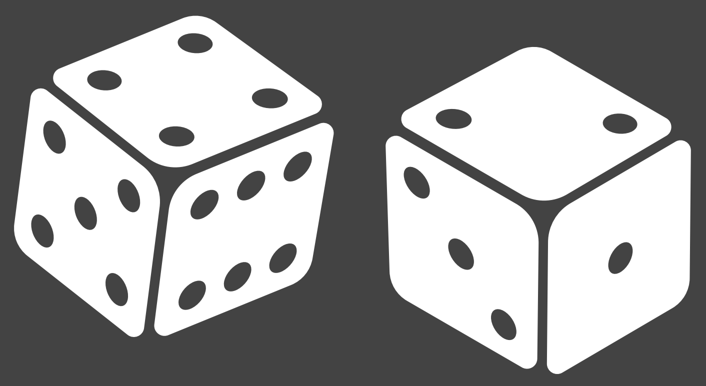
Theme I — Submodular Optimization

f(🍕 | 🥕) ≥ *f*(🍕 | 🥕 , 🍩)

Theme II — Stable Algorithms



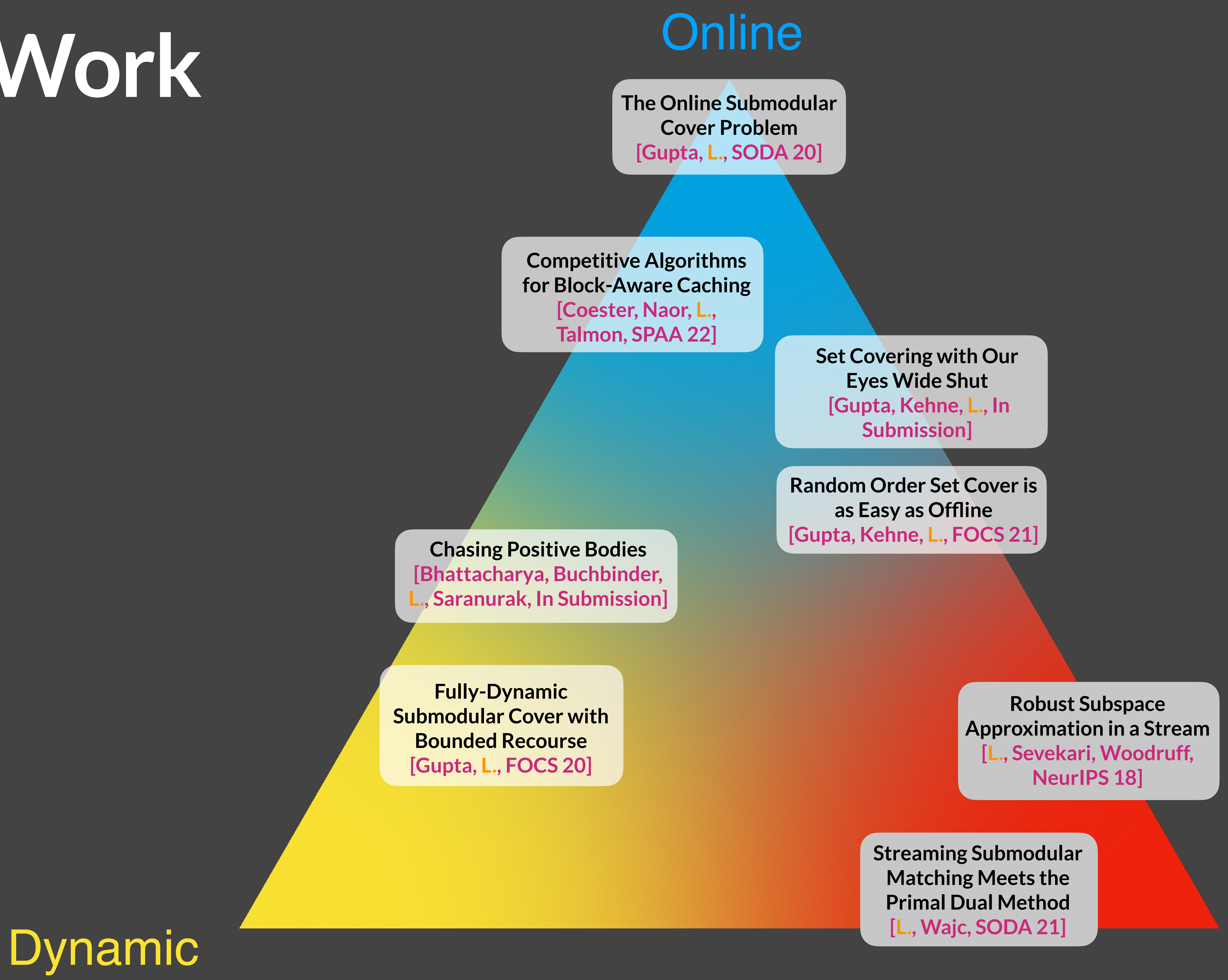
Theme III — Beyond Worst-Case Analysis



Conclusion

Conclusion

My Work



Finding Skewed Subcubes Under a Distribution [Gopalan, L., Wieder, ITCS 20]

FigureSeer: Parsing Result-Figures in Research Papers [Siegel, Horvitz, L., Divvala, Farhadi, ECCV 16]

Beyond Sentential Semantic Parsing: Tackling the Math SAT with a Cascade of Tree Transducers [Hopkins, Petrscu-Prahova, L., Le Bras, Herrasti, Joshi, EMNLP 17]

… and others in AI, ML, Fairness

Short/Medium Term Directions

Short/Medium Term Directions

1. Does LearnOrCover idea solve other problems?
   Do ideas transfer to random order Streaming?
   Unified theory of random order algorithms?

Short/Medium Term Directions

1. Does LearnOrCover idea solve other problems?
   Do ideas transfer to random order Streaming?
   Unified theory of random order algorithms?

2. Other “chaseable” constraint families, beyond mixed packing/covering? Stable clustering problems?

Short/Medium Term Directions

1. Does LearnOrCover idea solve other problems?
   Do ideas transfer to random order Streaming?
   Unified theory of random order algorithms?

2. Other “chaseable” constraint families, beyond mixed packing/covering? Stable clustering problems?
   (Big demand for this from industry!)

Short/Medium Term Directions

1. Does LearnOrCover idea solve other problems?
   Do ideas transfer to random order Streaming?
   Unified theory of random order algorithms?

2. Other "chaseable" constraint families, beyond mixed packing/covering? Stable clustering problems?
   (Big demand for this from industry!)

3. Do ideas work for update-time Dynamic algorithms?

Long Term Ambitions

Long Term Ambitions

1. Submodularity Under the Hood: can we get better algorithms by exploiting "submodular aspects" of non-submodular problems?

Long Term Ambitions

1. Submodularity Under the Hood: can we get better algorithms by exploiting "submodular aspects" of non-submodular problems?

2. Algorithms meet Data: How can we exploit things we learned yesterday? Beyond Bayesian/Stochastic models? Non-stationary generative processes?

Long Term Ambitions

1. Submodularity Under the Hood: can we get better algorithms by exploiting “submodular aspects” of non-submodular problems?

2. Algorithms meet Data: How can we exploit things we learned yesterday? Beyond Bayesian/Stochastic models? Non-stationary generative processes?

3. Apply Theory in Practice: Does my work inform useful heuristics? New collaborations on real world applications?

Research Philosophy

Research Philosophy

1. Simplicity: better in practice & easier to explain.

Research Philosophy

1. Simplicity: better in practice & easier to explain.

2. Abstraction: gets at deep principle explaining a phenomenon & automatically yields many applications.

Research Philosophy

1. Simplicity: better in practice & easier to explain.

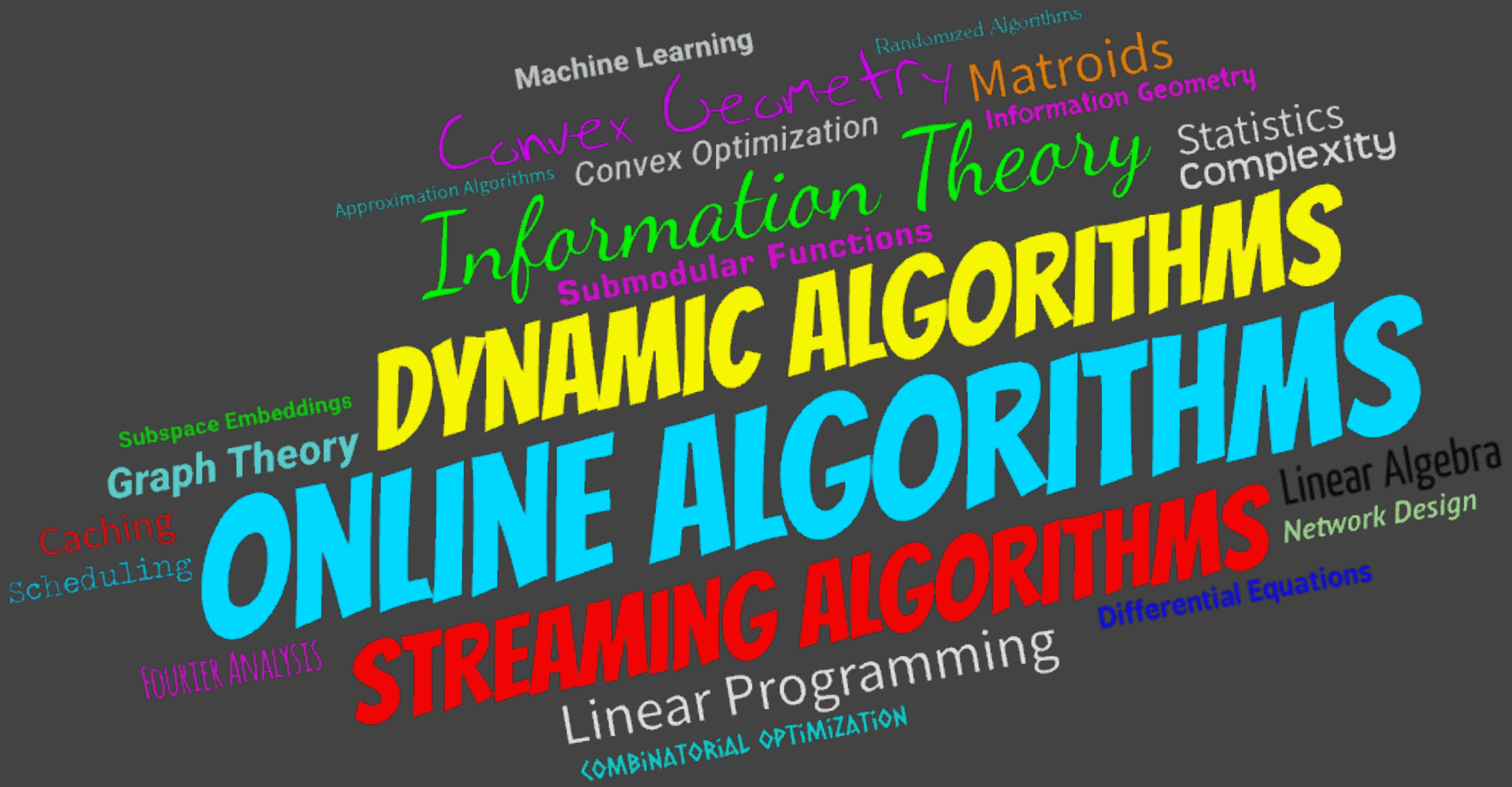
2. Abstraction: gets at deep principle explaining a phenomenon & automatically yields many applications.

3. Practical Impact: stay anchored to needs of real world & plentiful source of inspiration.

Algorithms & Uncertainty

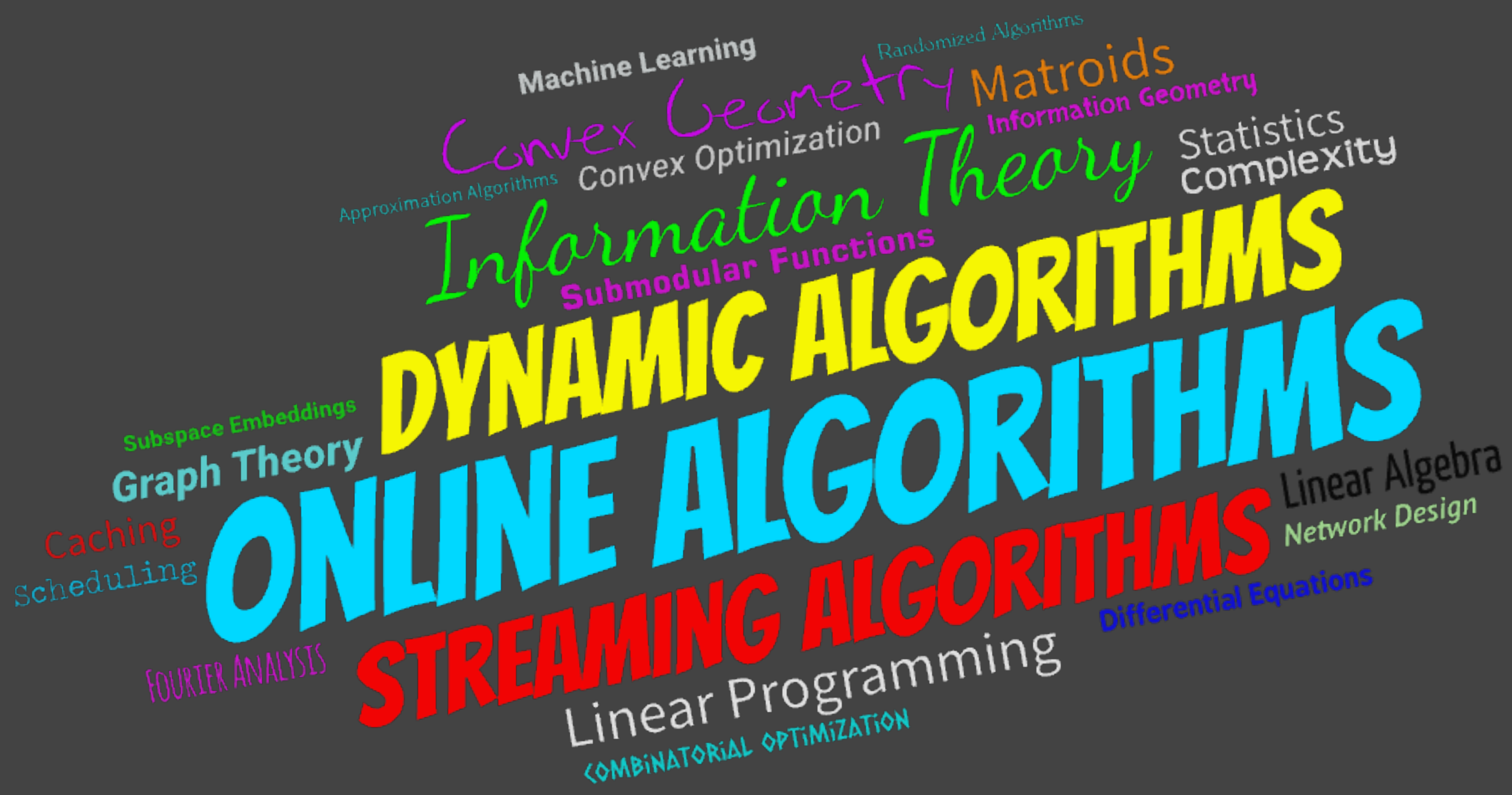
Algorithms & Uncertainty

- **Intersection of many beautiful branches of CS & Math!**



Algorithms & Uncertainty

- **Intersection** of many beautiful branches of **CS** & **Math**!

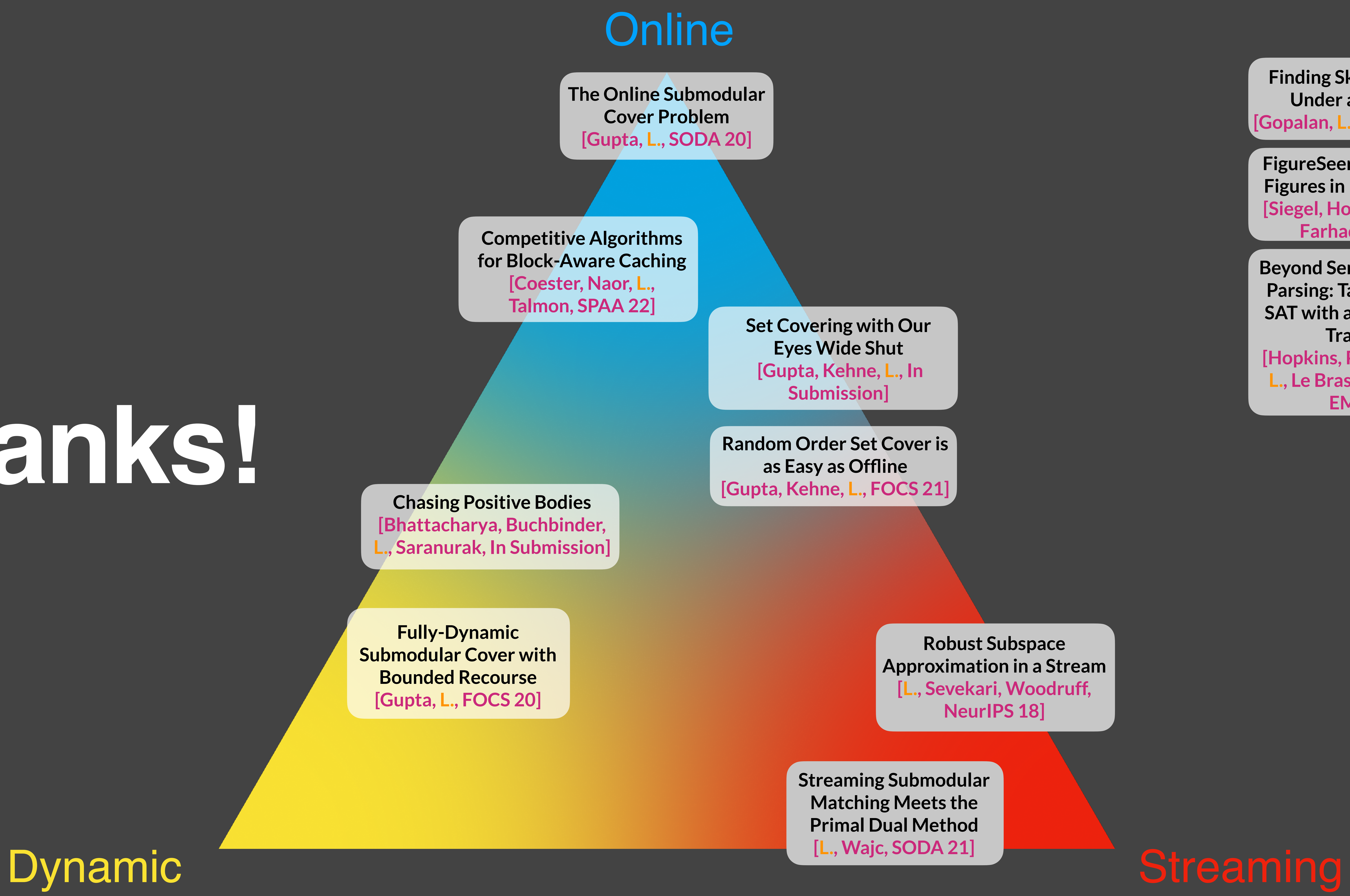


- **Fun** & **approachable** on-ramp to research!

Recent/Current Collaborators

- **Carnegie Mellon University**: Anupam Gupta, Anish Sevekari, David Woodruff
- **Harvard**: Gregory Kehne
- **U Michigan**: Thatchaphol Saranurak
- **Duke**: Debmalya Panigrahi
- **Tel Aviv University**: Niv Buchbinder, Haim Kaplan, Yaniv Sadeh
- **Technion**: Seffi Naor, Ohad Talmon, David Naori
- **Oxford**: Christian Coester
- **University of Warwick**: Sayan Bhattacharya
- **London School of Economics**: Neil Olver, Franziska Eberle
- **University of Bremen**: Nicole Megow
- **Google Research**: Ravi Kumar, Rajesh Jayaram, David Wajc
- **Apple**: Parikshit Gopalan
- **VMWare**: Udi Wieder

Thanks!



Finding Skewed Subcubes Under a Distribution [Gopalan, L., Wieder, ITCS 20]

FigureSeer: Parsing Result-Figures in Research Papers [Siegel, Horvitz, L., Divvala, Farhadi, ECCV 16]

Beyond Sentential Semantic Parsing: Tackling the Math SAT with a Cascade of Tree Transducers [Hopkins, Petrscu-Prahova, L., Le Bras, Herrasti, Joshi, EMNLP 17]