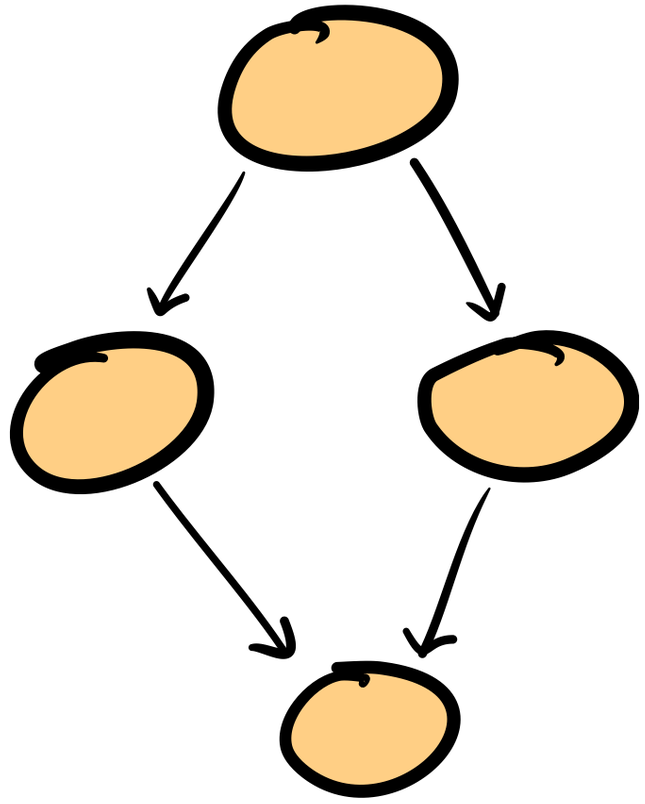
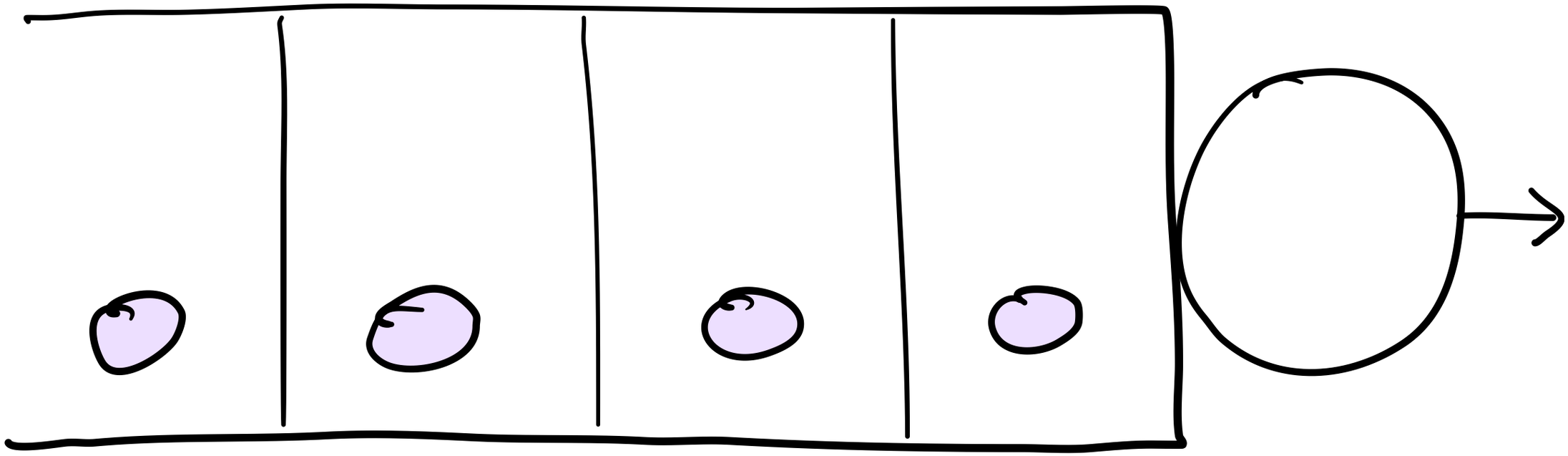
Optimally Scheduling Jobs with Multiple Tasks

Ziv Scully
Guy Blelloch
Mor Harchol-Balter
Alan Scheller-Wolf

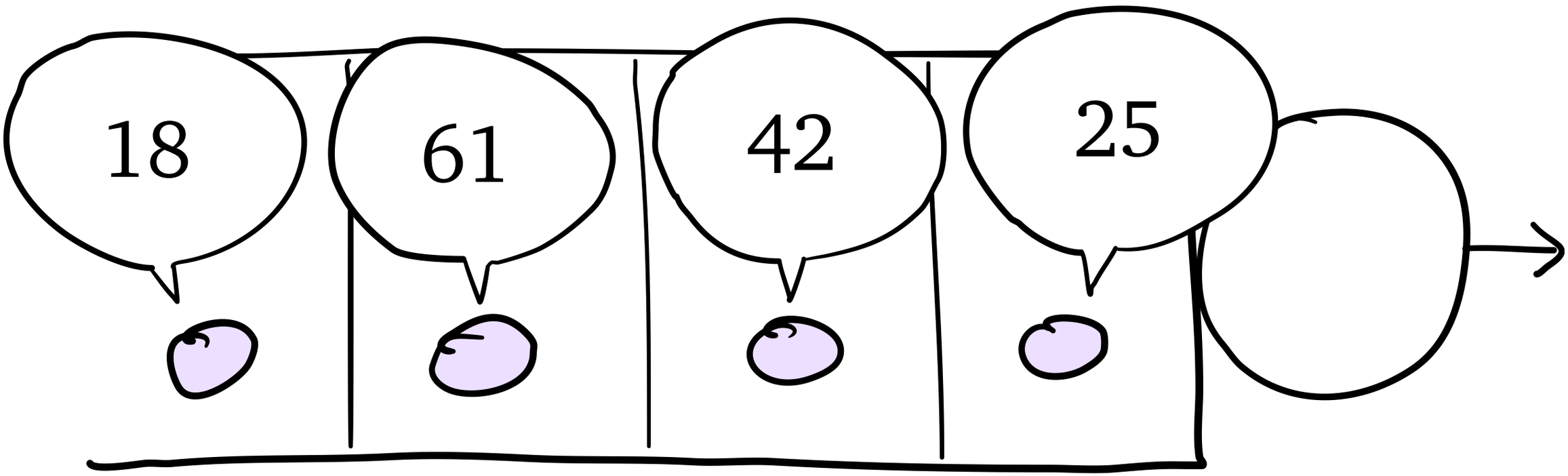
Carnegie Mellon University



Classic: Multiclass M/G/1

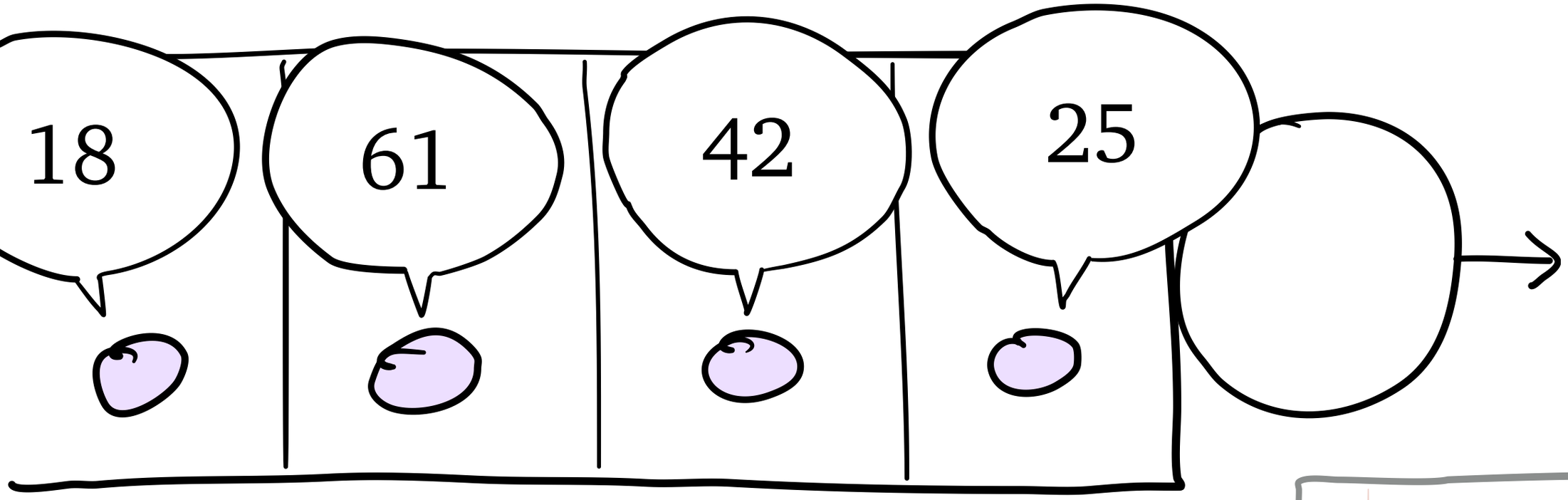


Classic: Multiclass M/G/1



Known sizes:

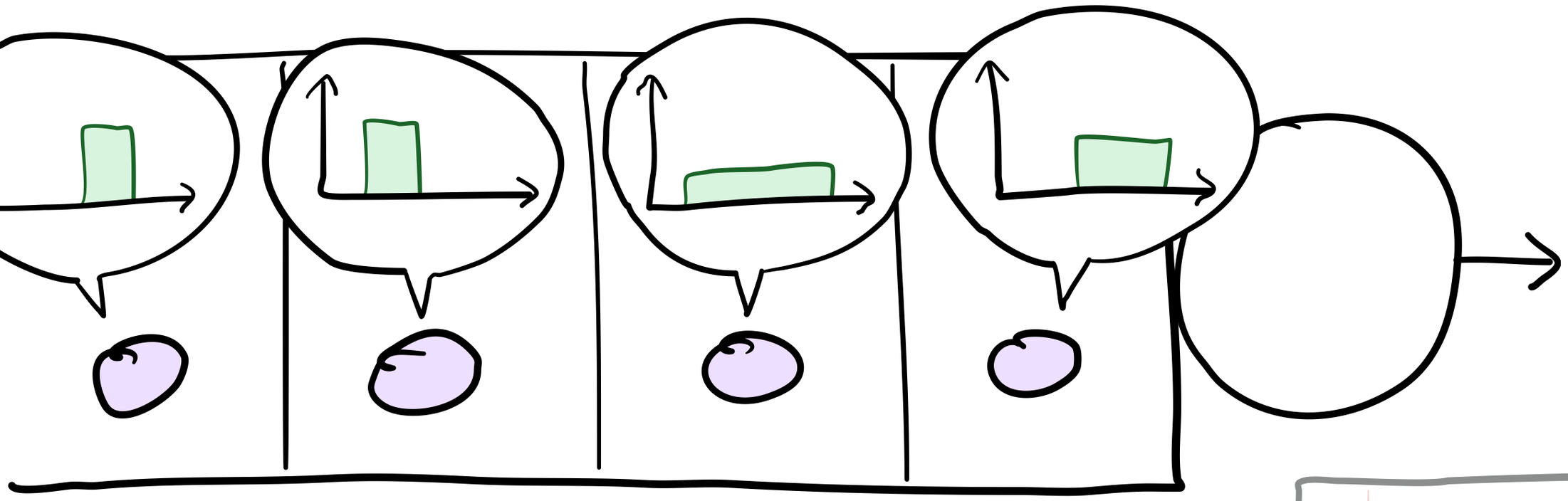
Classic: Multiclass M/G/1



Known sizes: **SRPT**

Shortest
Remaining
Processing
Time

Classic: Multiclass M/G/1

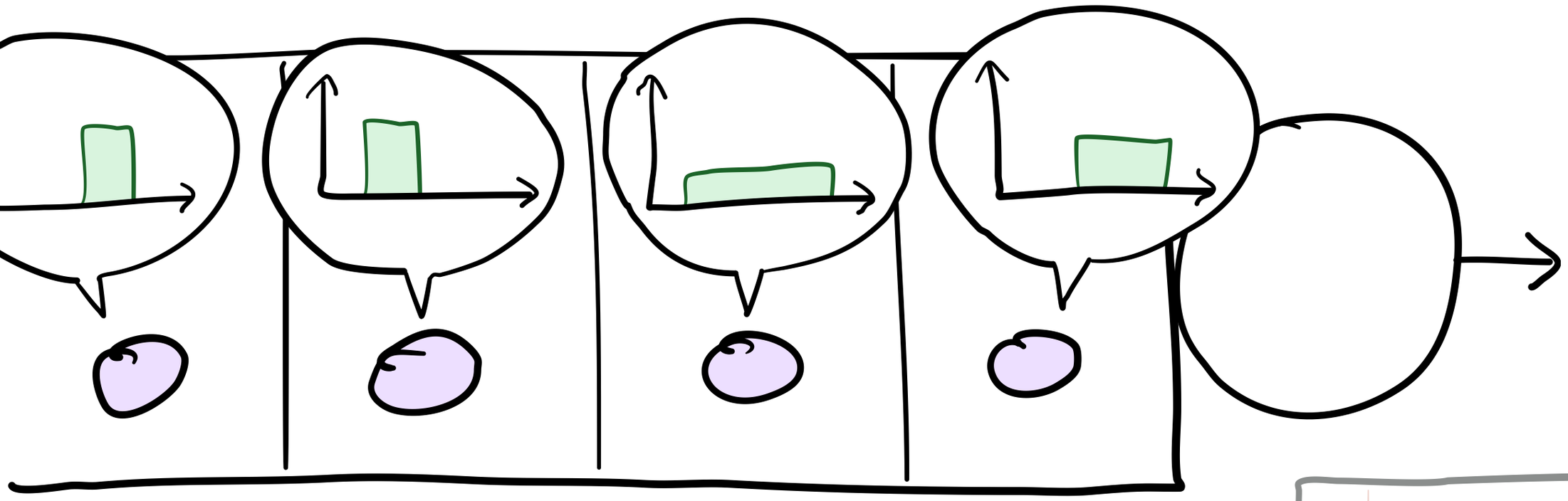


Known sizes: **SRPT**

Uniform distributions:

Shortest
Remaining
Processing
Time

Classic: Multiclass M/G/1

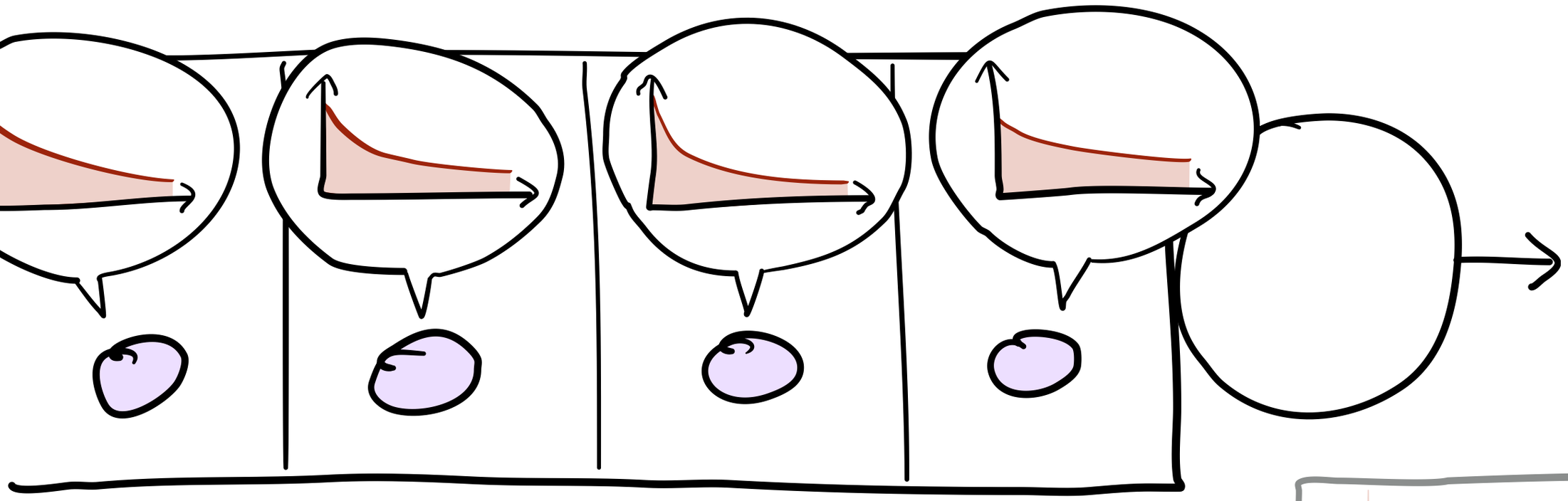


Known sizes: **SRPT**

Uniform distributions: **SERPT**

Shortest
Expected
Remaining
Processing
Time

Classic: Multiclass M/G/1



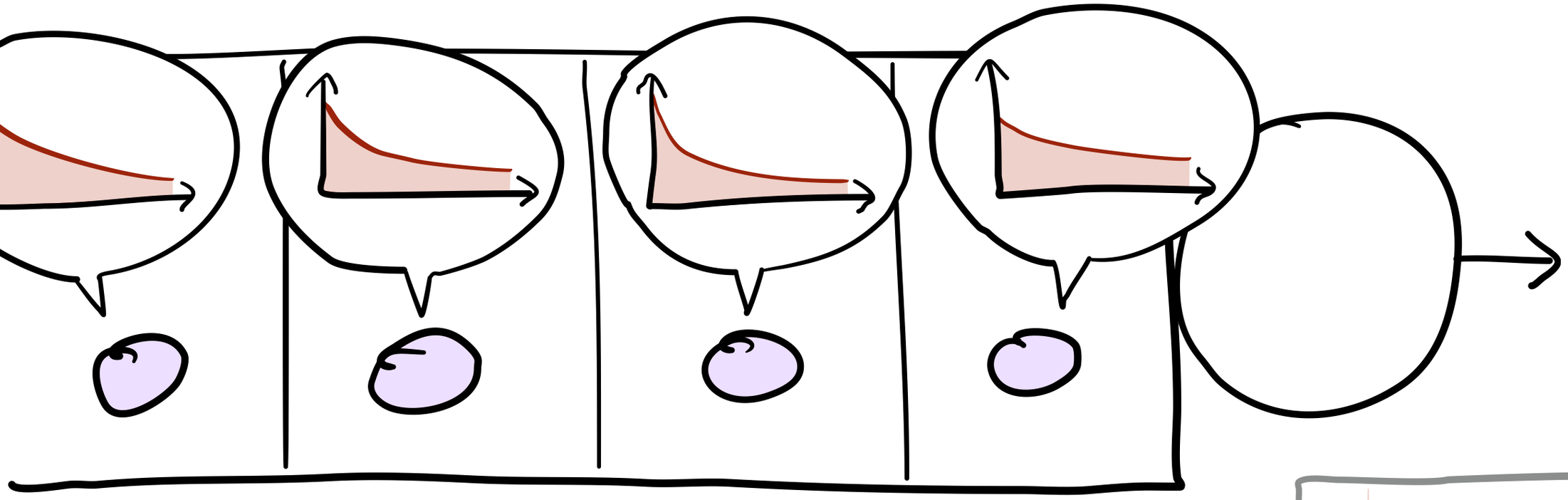
Known sizes: **SRPT**

Uniform distributions: **SERPT**

Pareto distributions:

Shortest
Expected
Remaining
Processing
Time

Classic: Multiclass M/G/1



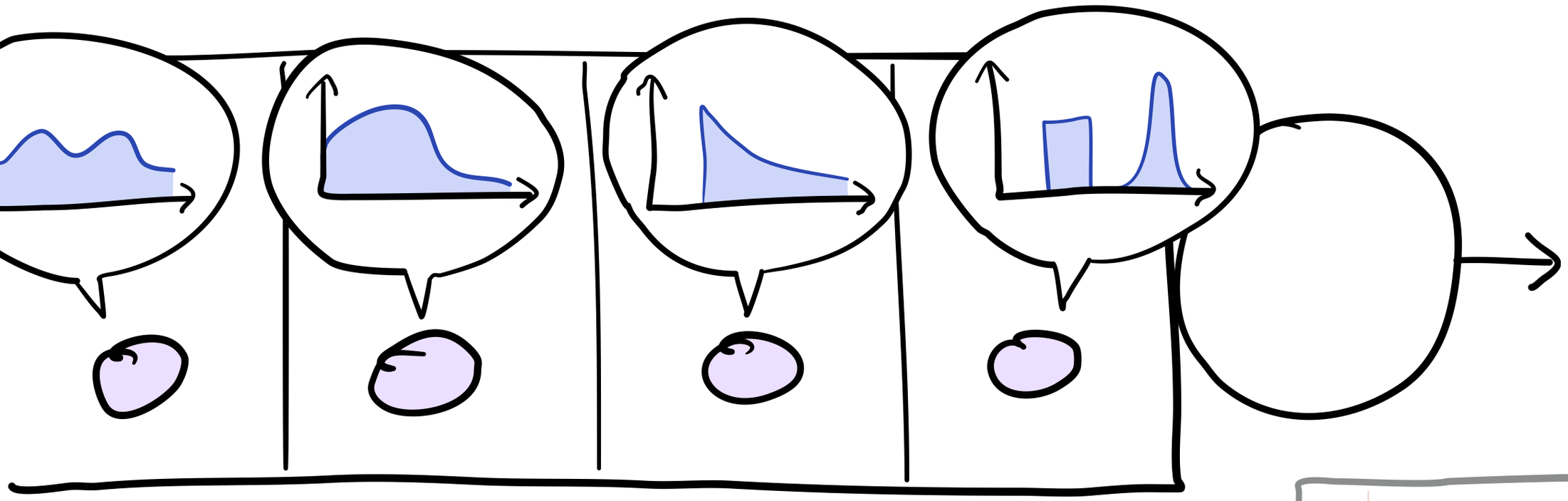
Known sizes: **SRPT**

Uniform distributions: **SERPT**

Pareto distributions: **HHR** (not SERPT!)

Highest
Hazard
Rate

Classic: Multiclass M/G/1



Known sizes: **SRPT**

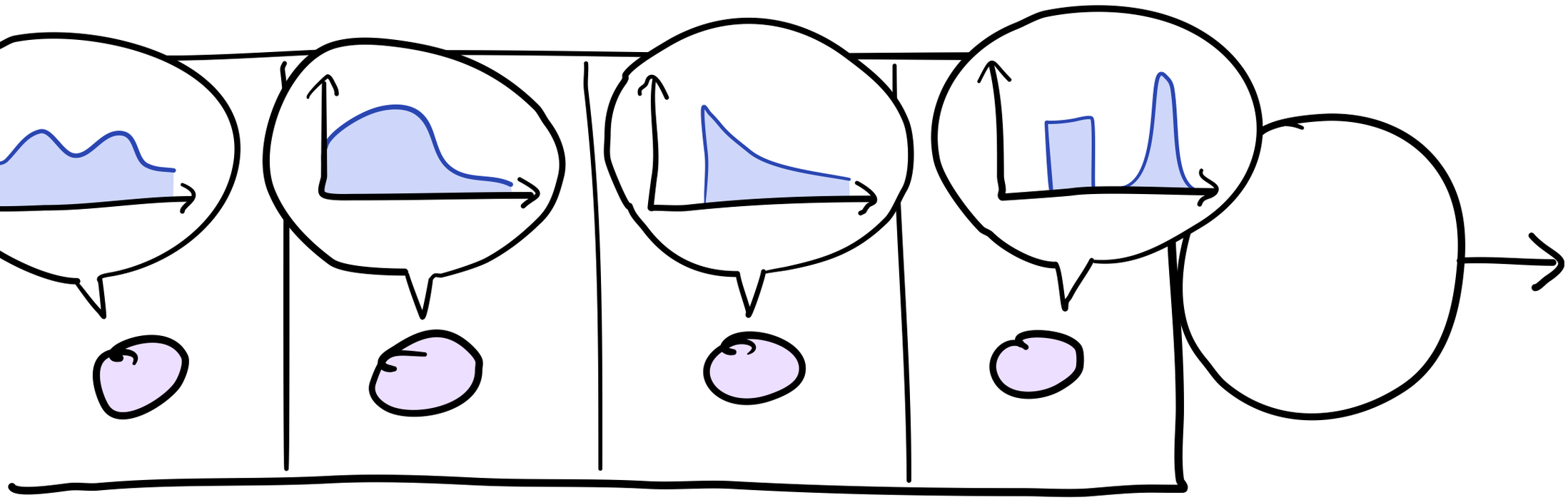
Uniform distributions: **SERPT**

Pareto distributions: **HHR** (not SERPT!)

General distributions:

Highest
Hazard
Rate

Classic: Multiclass M/G/1



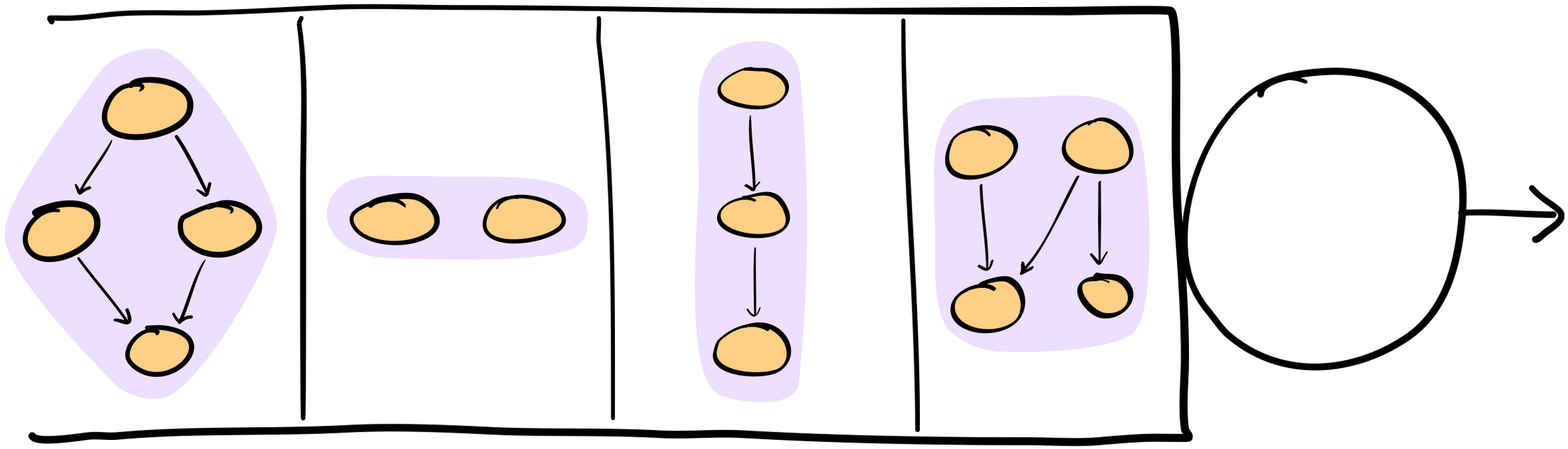
Known sizes: **SRPT**

Uniform distributions: **SERPT**

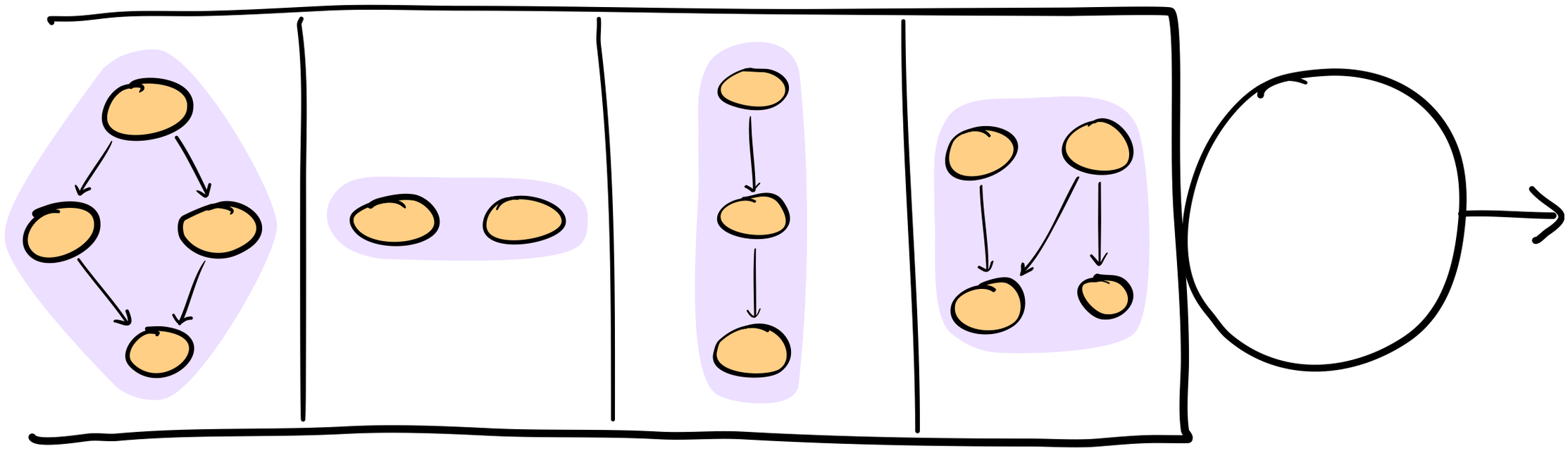
Pareto distributions: **HHR** (not SERPT!)

General distributions: **Gittins index** policy

Today: Multitask Jobs

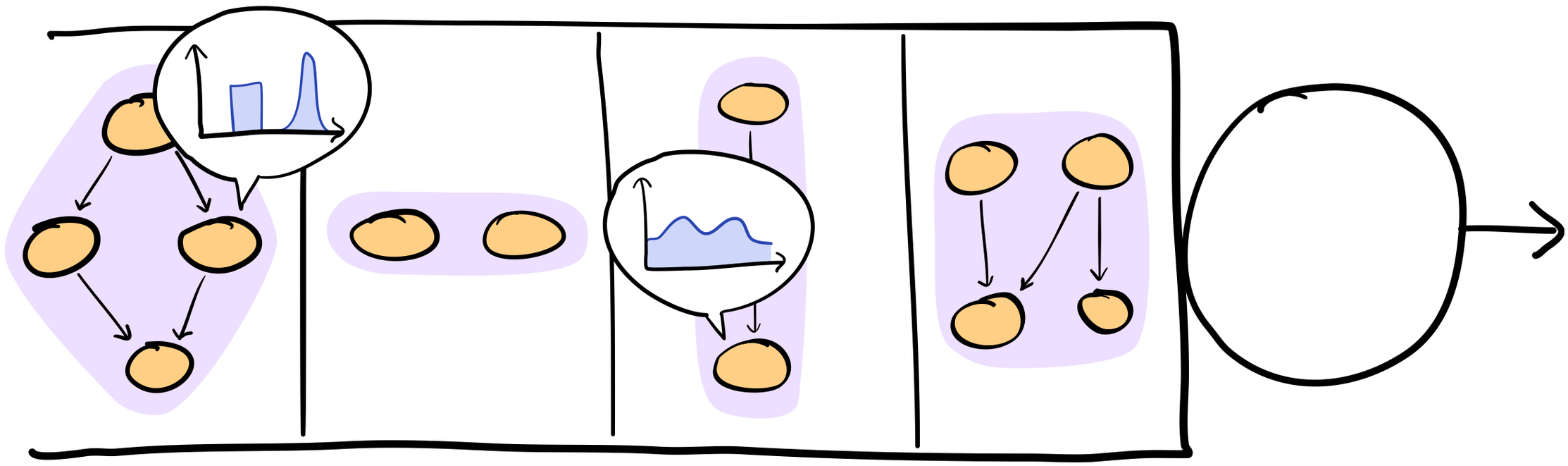


Today: Multitask Jobs



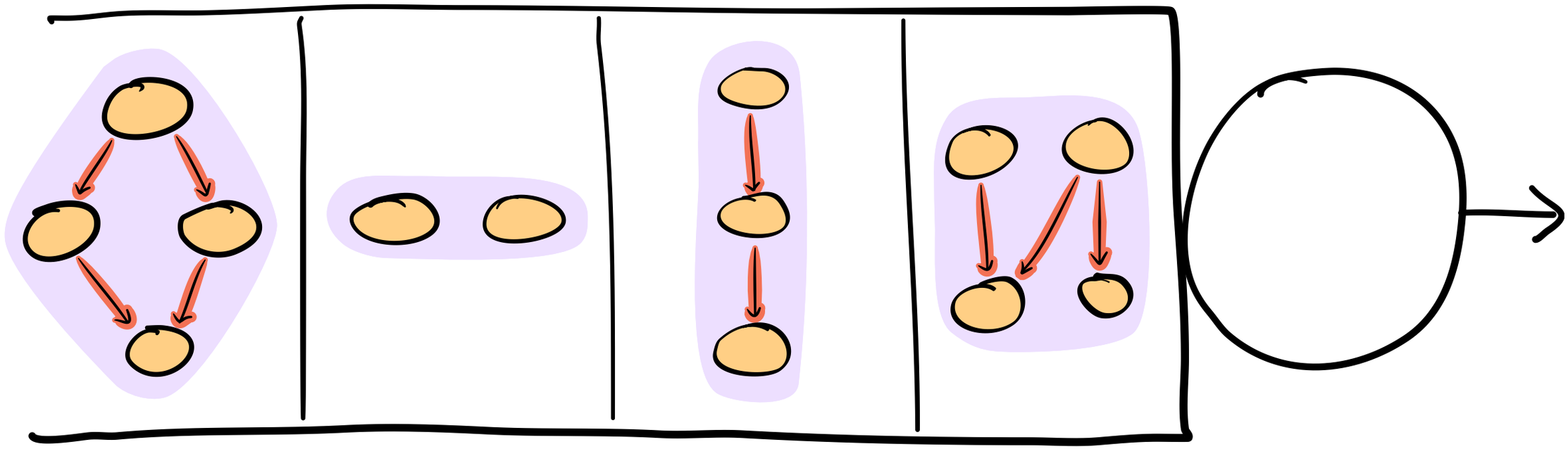
Jobs consist of multiple **tasks** in a DAG

Today: Multitask Jobs



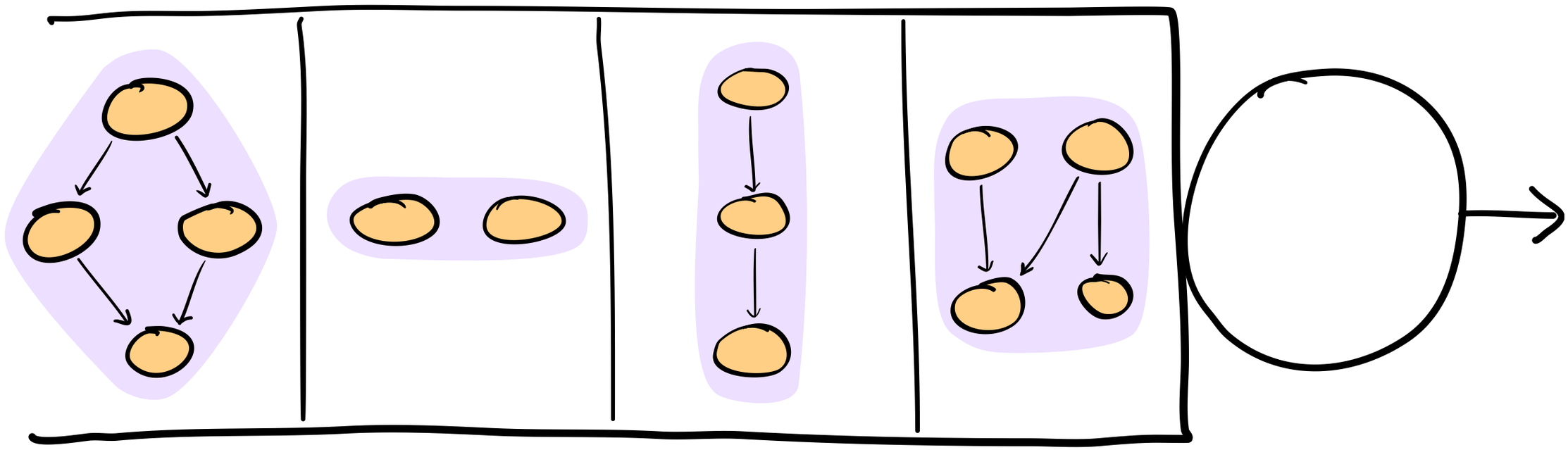
Jobs consist of multiple **tasks** in a DAG

Today: Multitask Jobs



Jobs consist of multiple **tasks** in a DAG

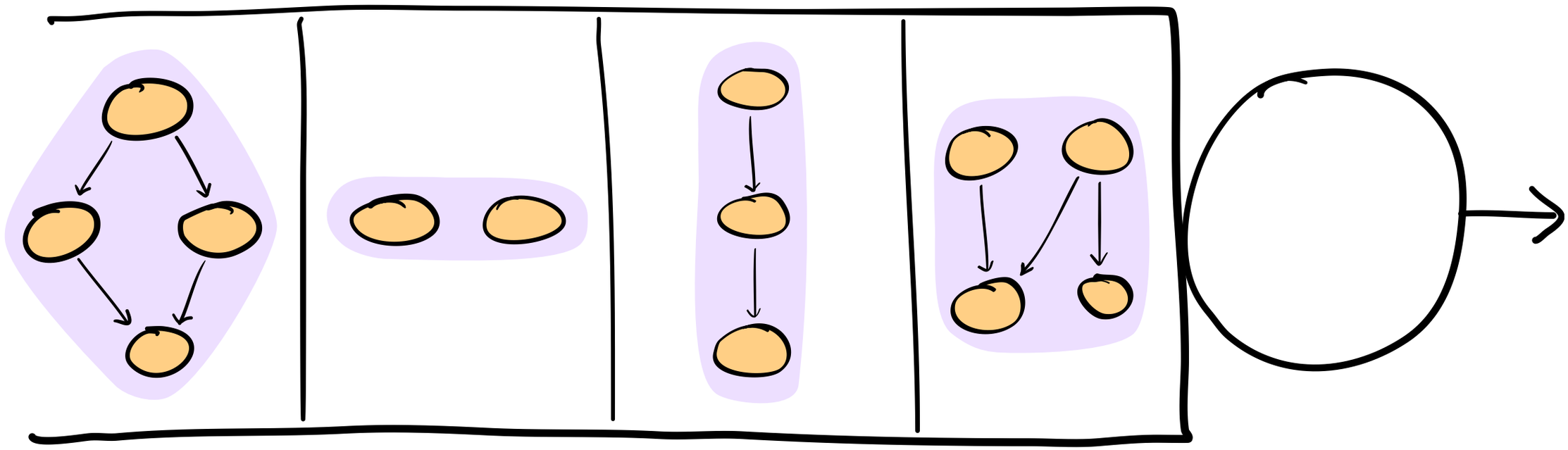
Today: Multitask Jobs



Jobs consist of multiple **tasks** in a DAG

Job not done until *all* tasks done

Today: Multitask Jobs



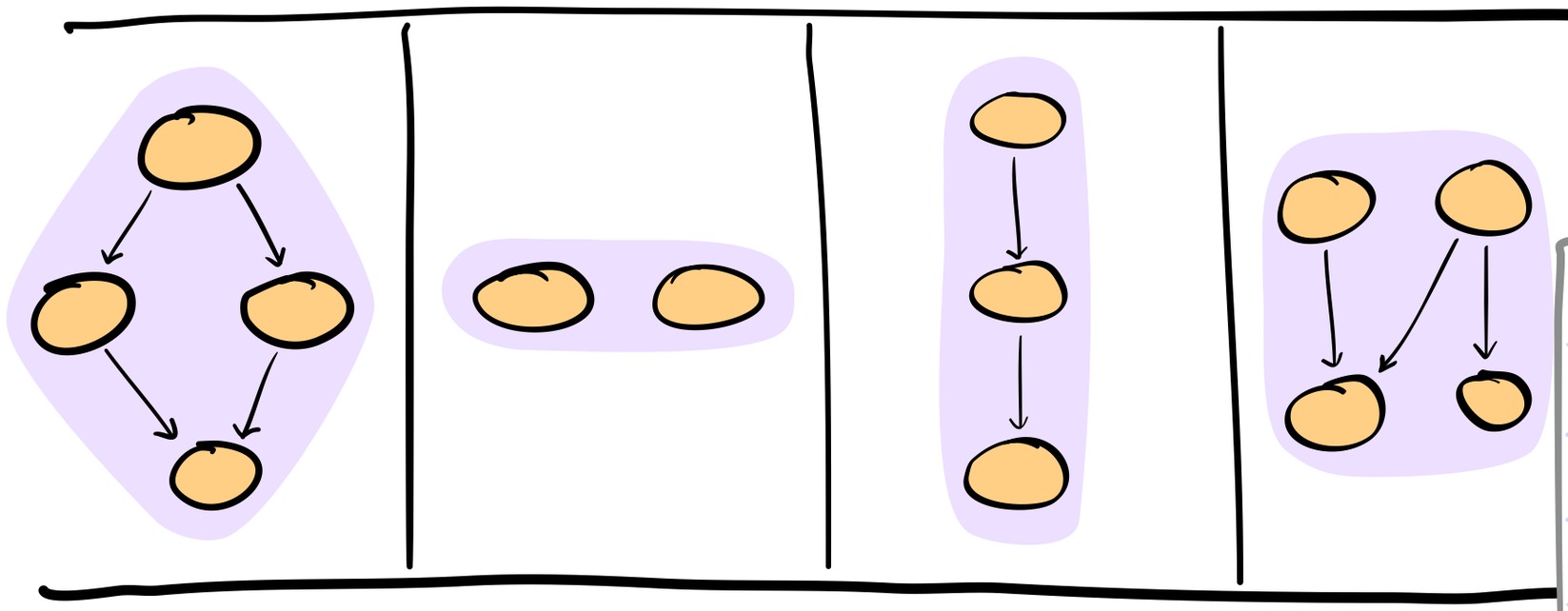
Jobs consist of multiple **tasks** in a DAG

Job not done until *all* tasks done

Goal: minimize mean response time of **jobs**

- Which **job**?
- Which **task**?

Today: Multitask Jobs



Serve tasks, not jobs
Preemption allowed
Single processor
No arrivals

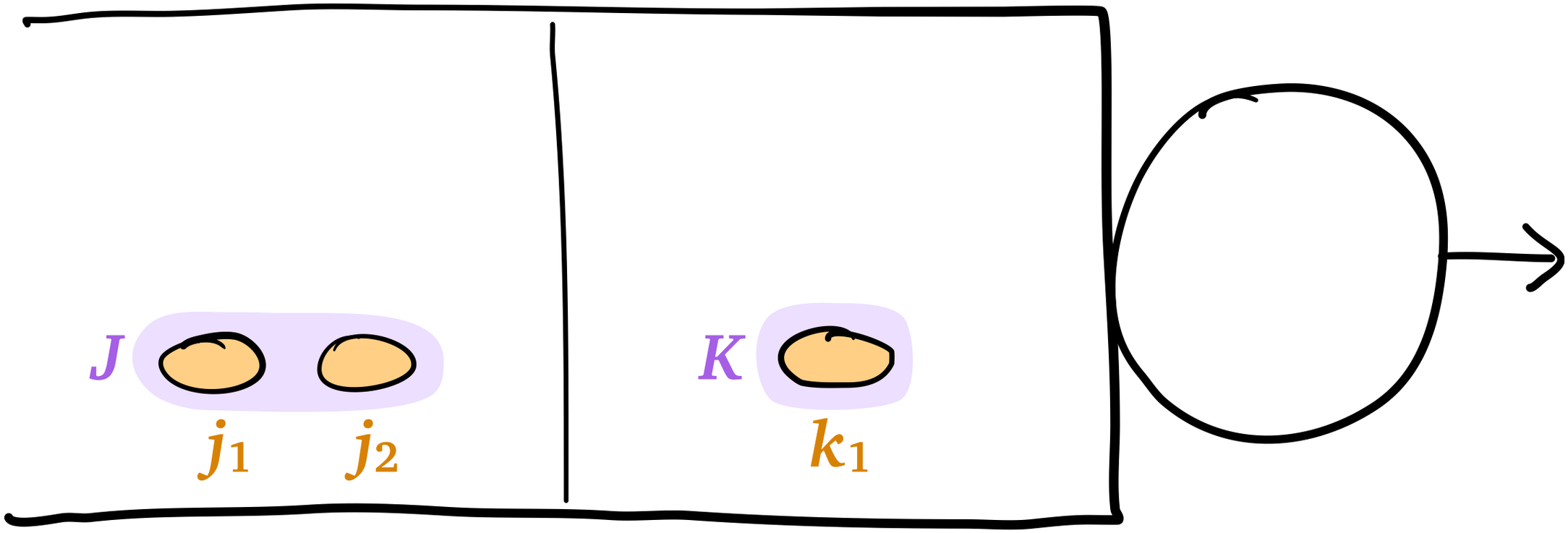
Jobs consist of multiple **tasks** in a DAG

Job not done until *all* tasks done

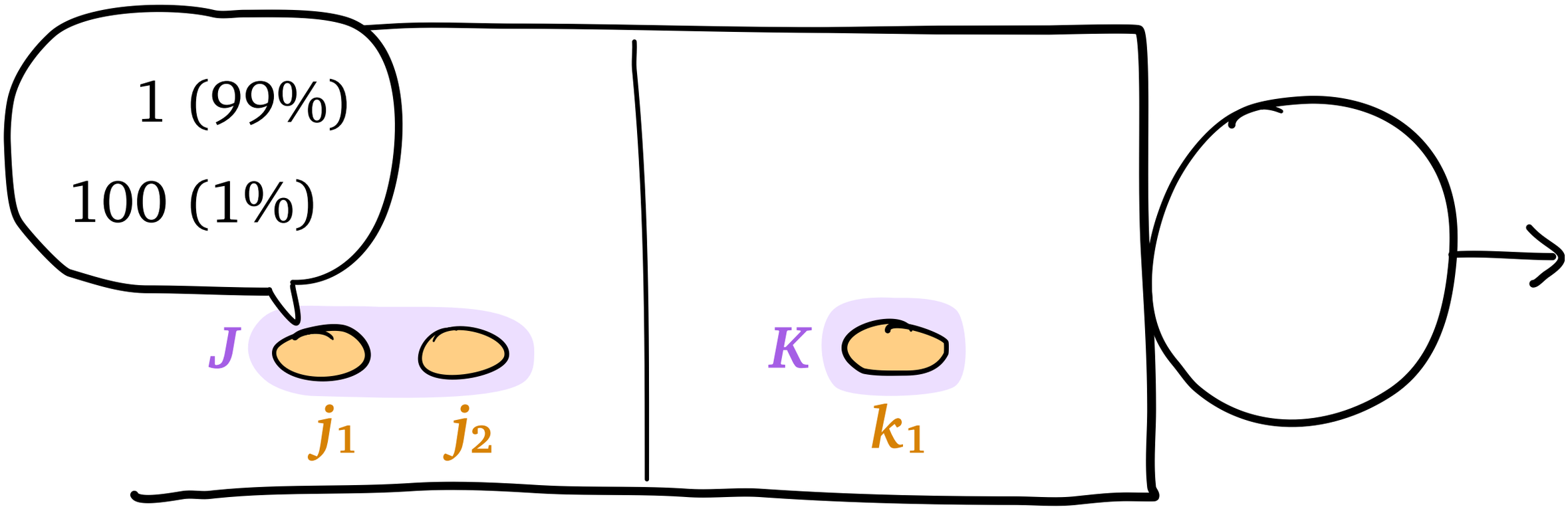
Goal: minimize mean response time of **jobs**

- Which **job**?
- Which **task**?

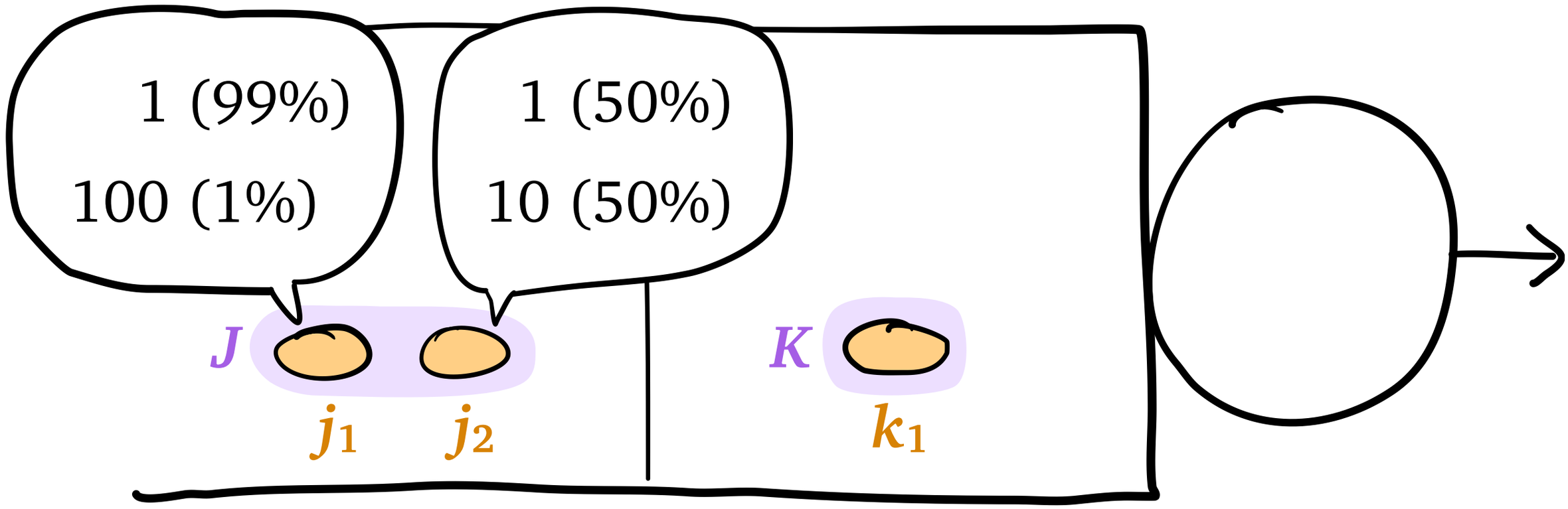
Example



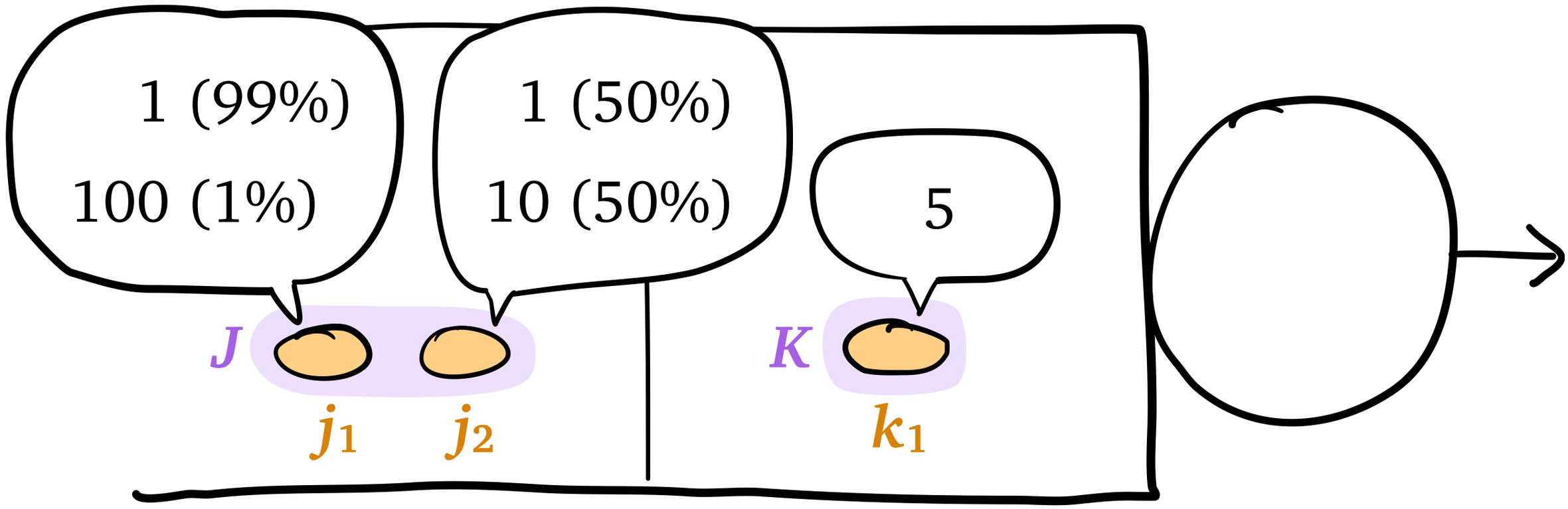
Example



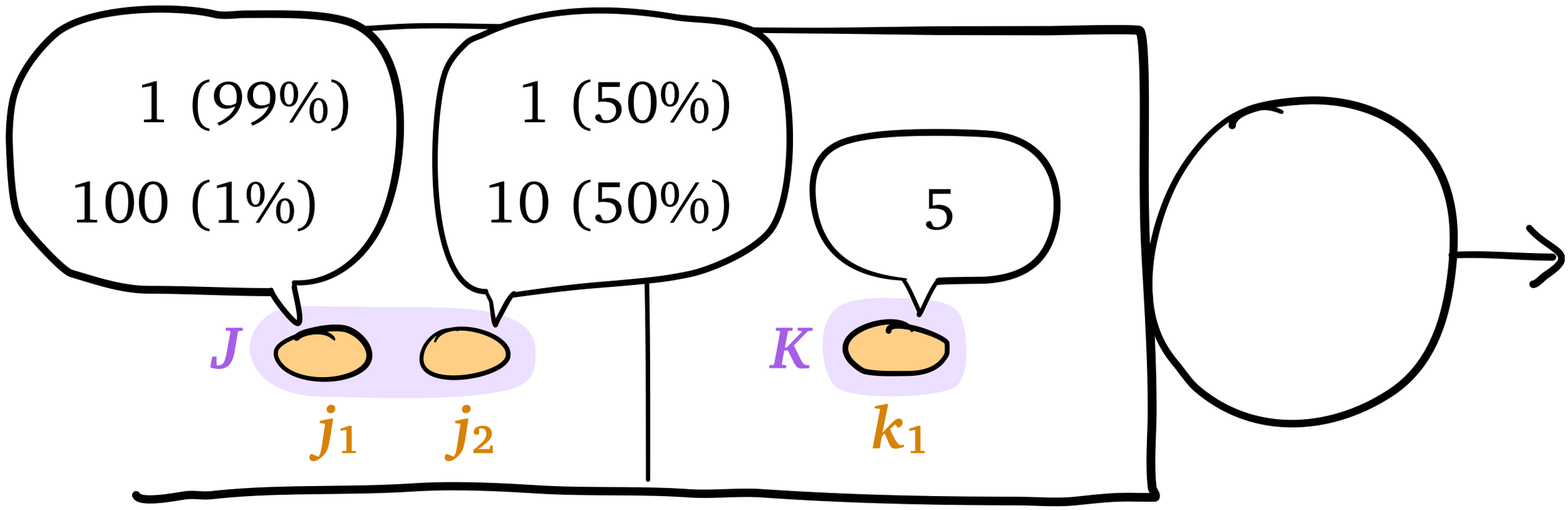
Example



Example

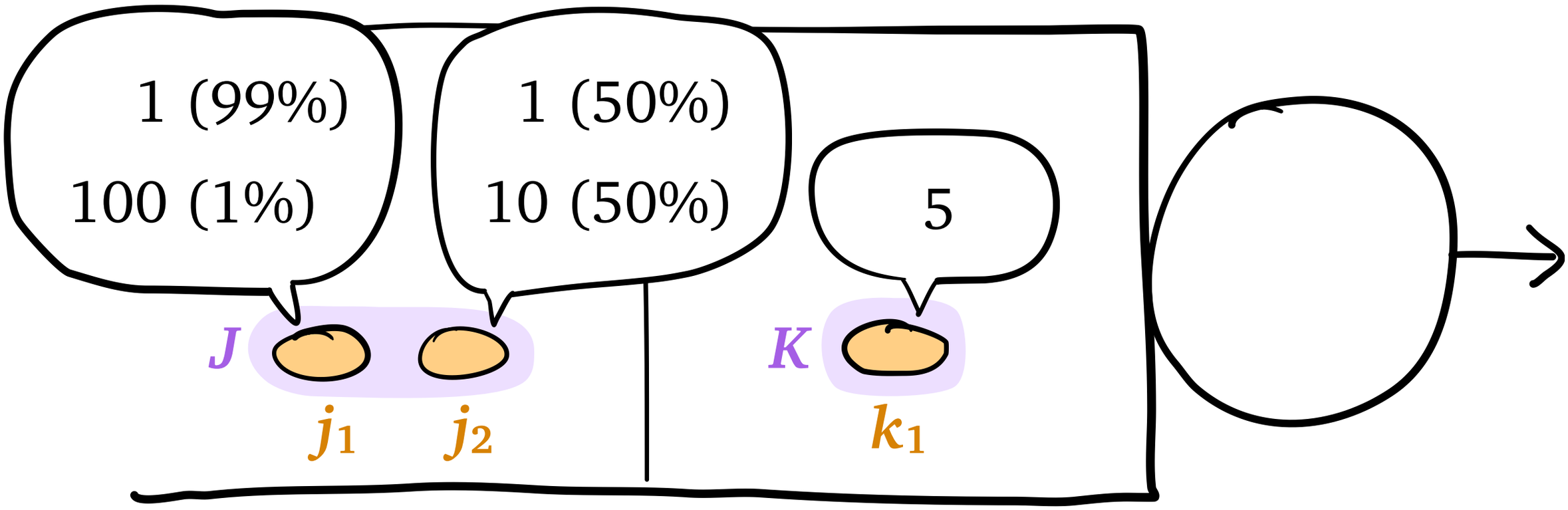


Example



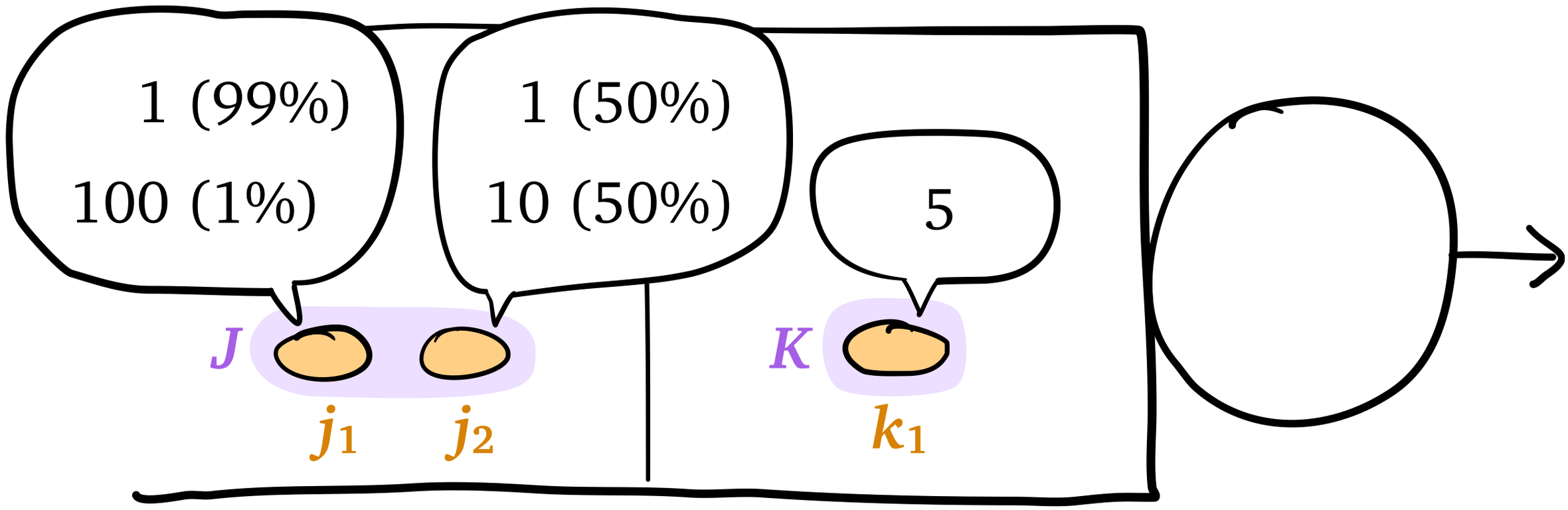
Which **job** should we run first?

Example



Which **job** should we run first? *J*

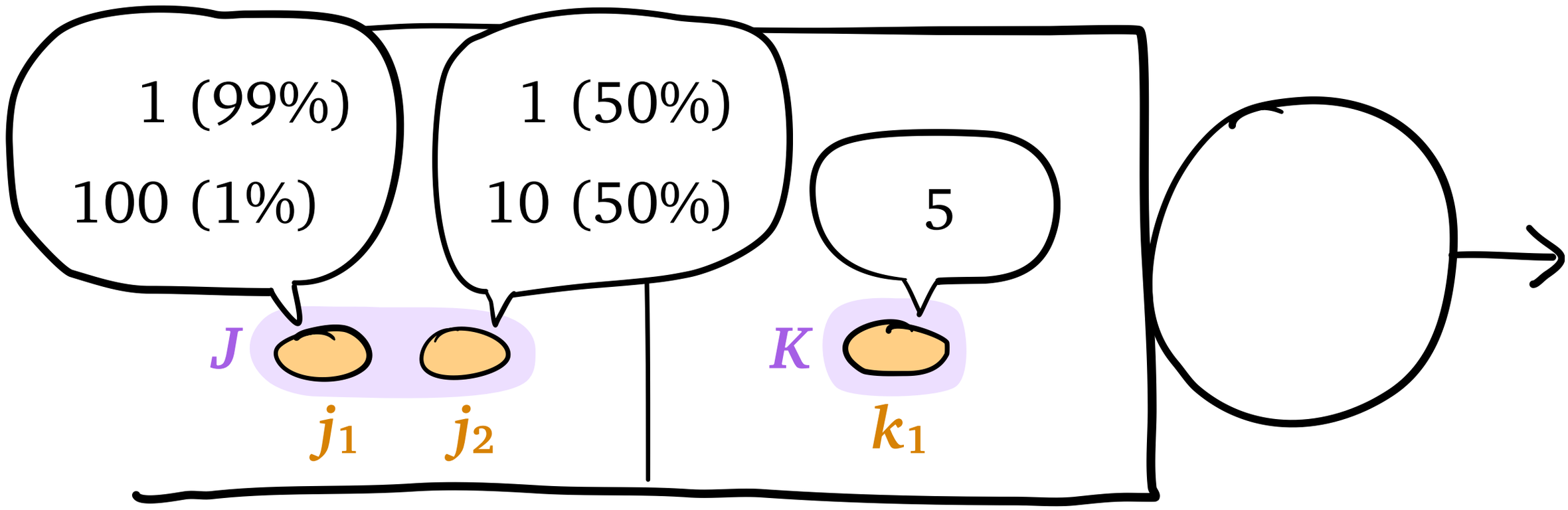
Example



Which **job** should we run first? *J*

Which **task** of *J* should we serve first?

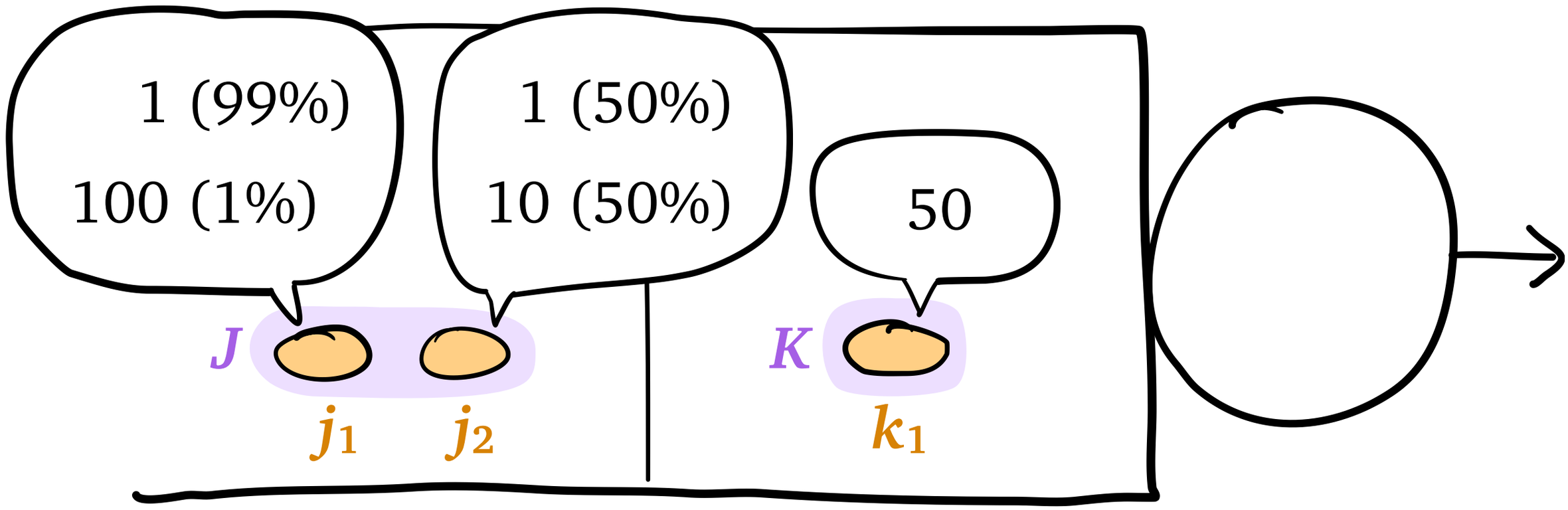
Example



Which **job** should we run first? J

Which **task** of J should we serve first? j_2

Example

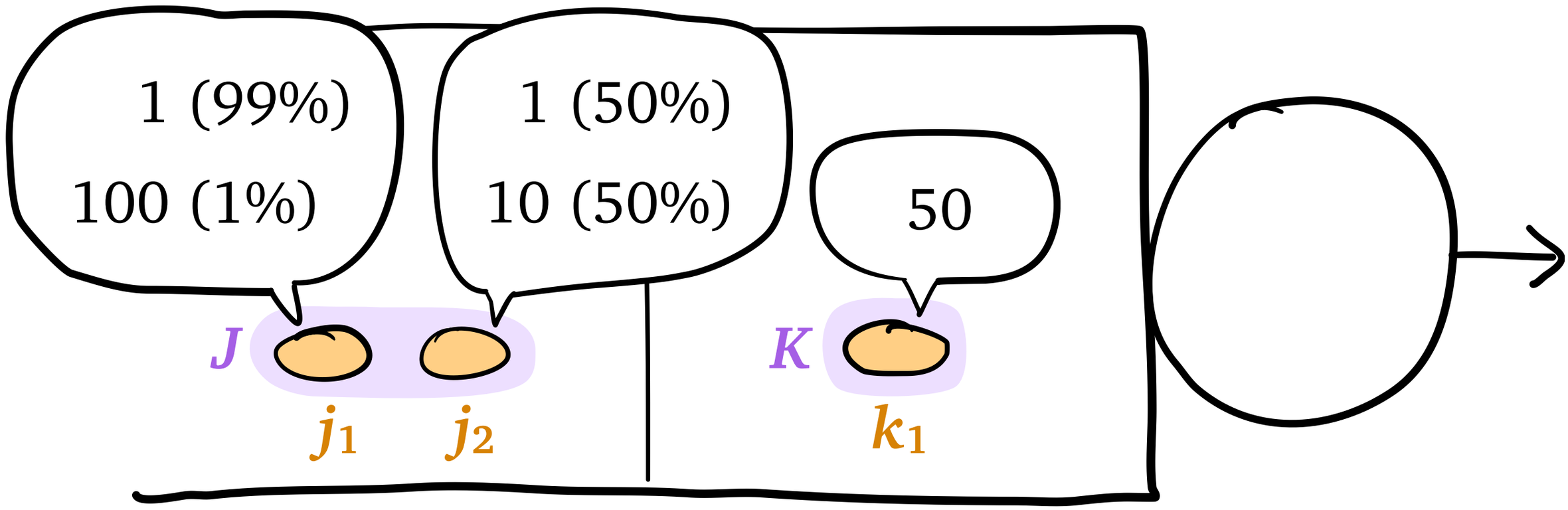


Which **job** should we run first? J

Which **task** of J should we serve first? j_2

What if we increase size of K to 50?

Example

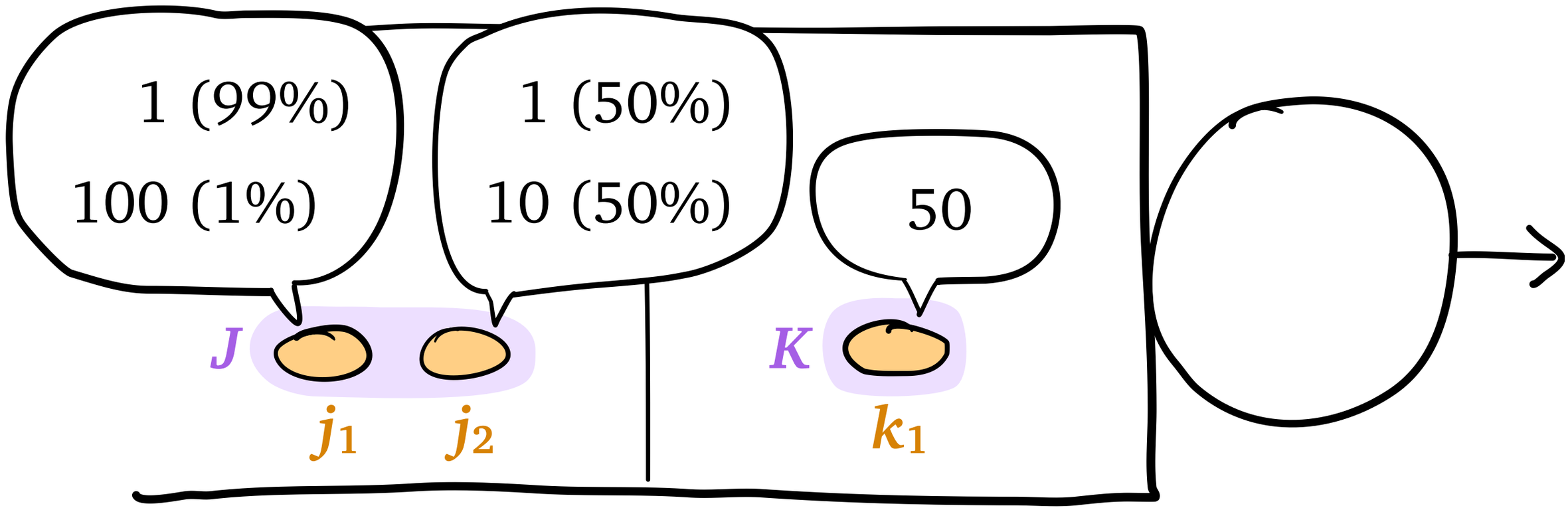


Which **job** should we run first? J

Which **task** of J should we serve first? j_2

What if we increase size of K to 50? j_1

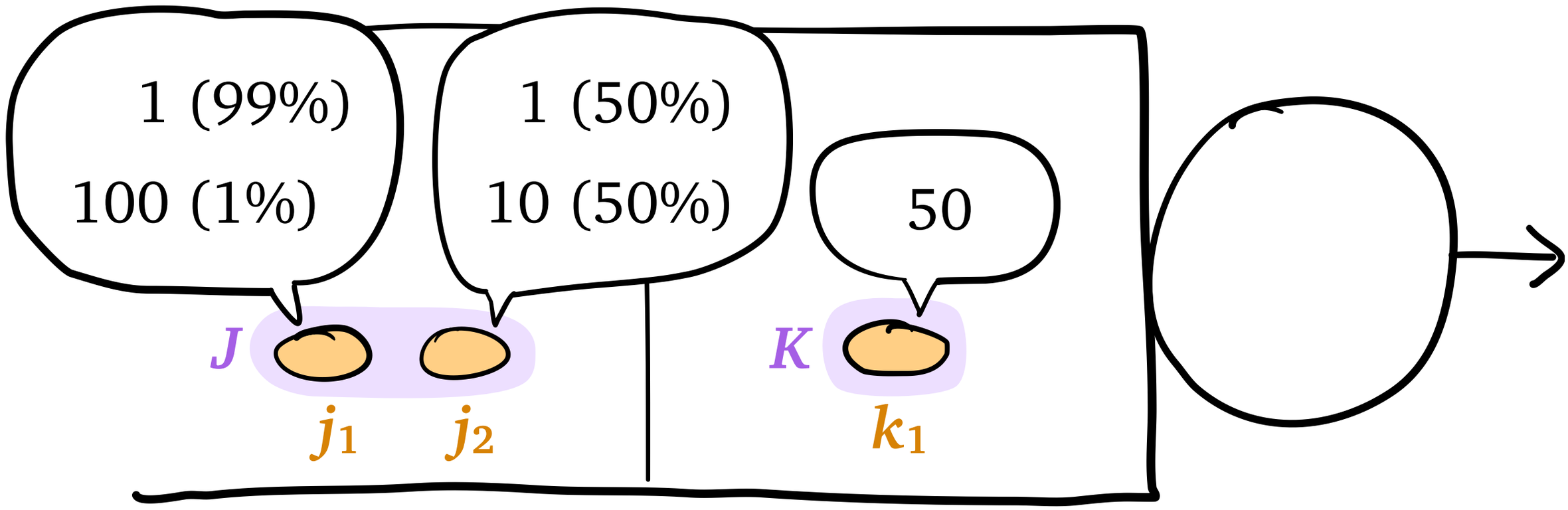
Example



Takeaways:

- Which **job**?
- Which **task**?

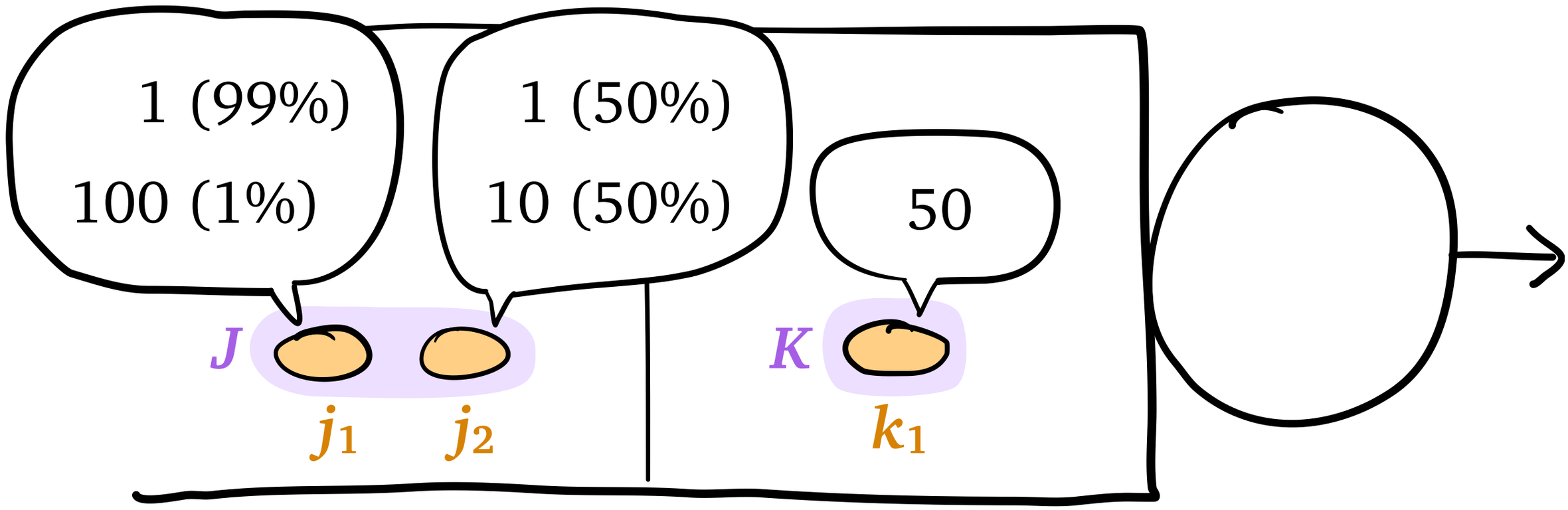
Example



Takeaways:

- Which **job**? *Not SERPT!*
- Which **task**?

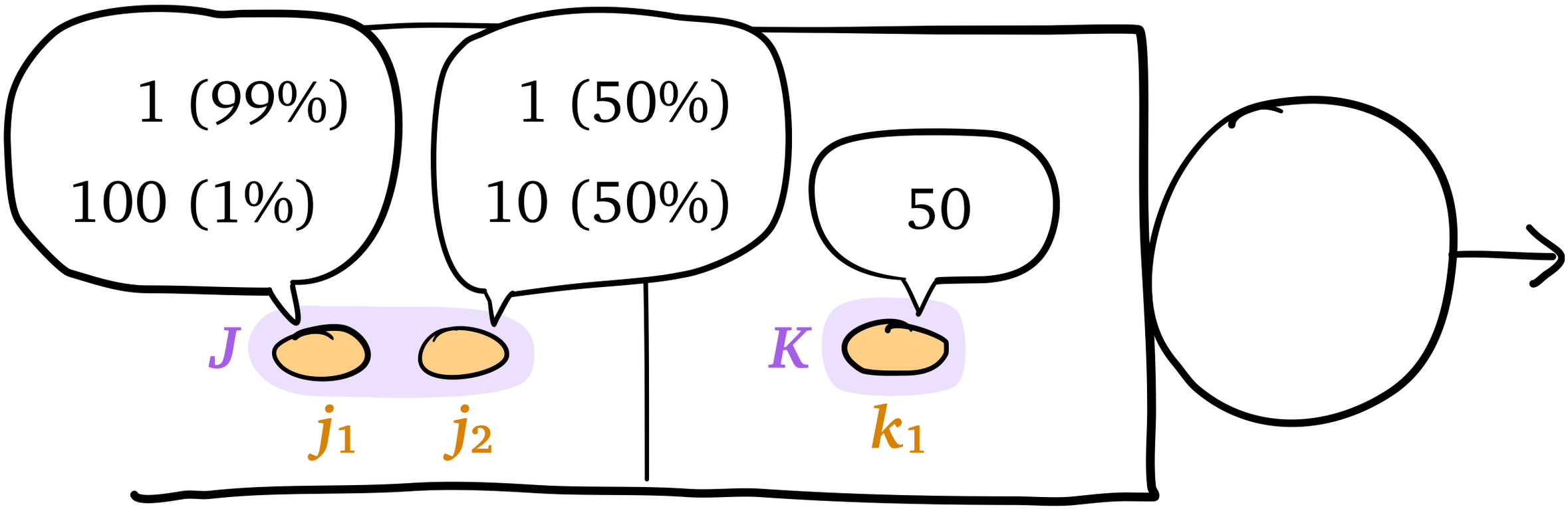
Example



Takeaways:

- Which **job**? *Not SERPT!*
- Which **task**? *"Most informative" first*

Example

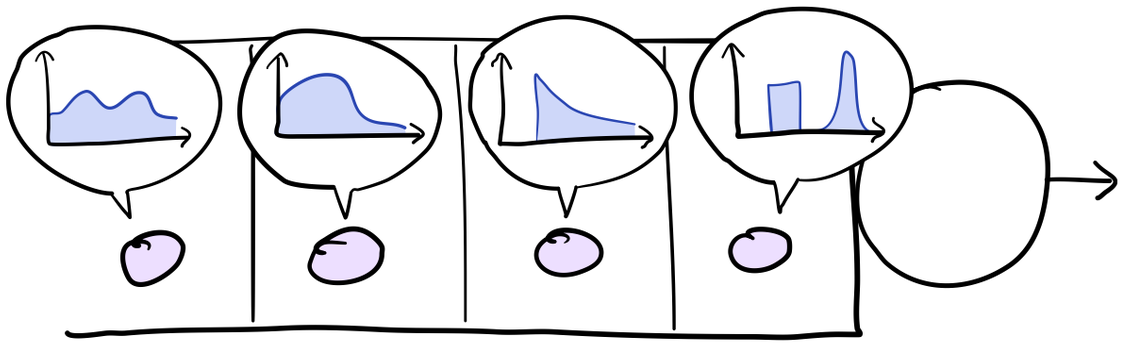


Takeaways:

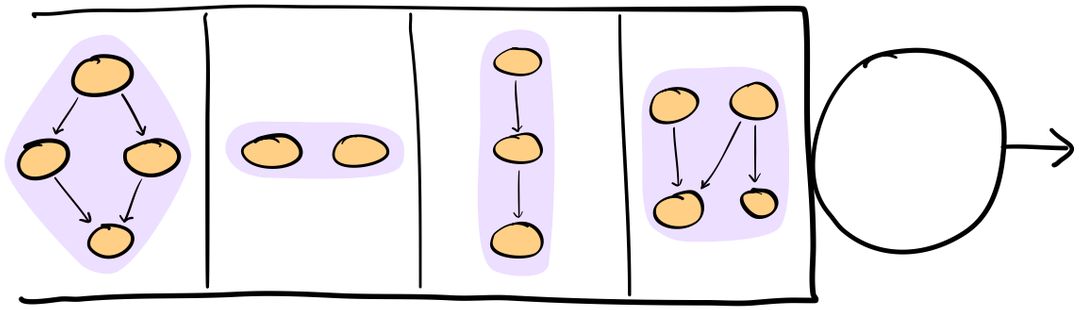
- Which **job**? *Not SERPT!*
- Which **task**? *"Most informative" first*

⚠ "Most informative" is context-dependent!

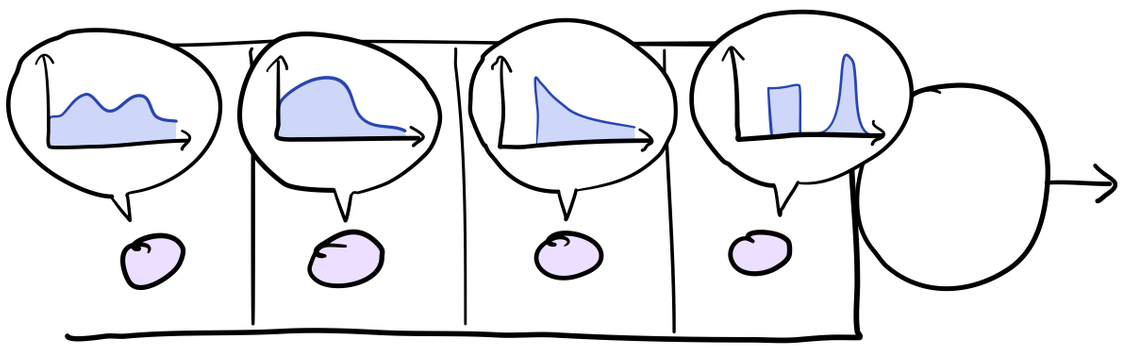
Prior Work: Single-Task



New Problem: Multitask

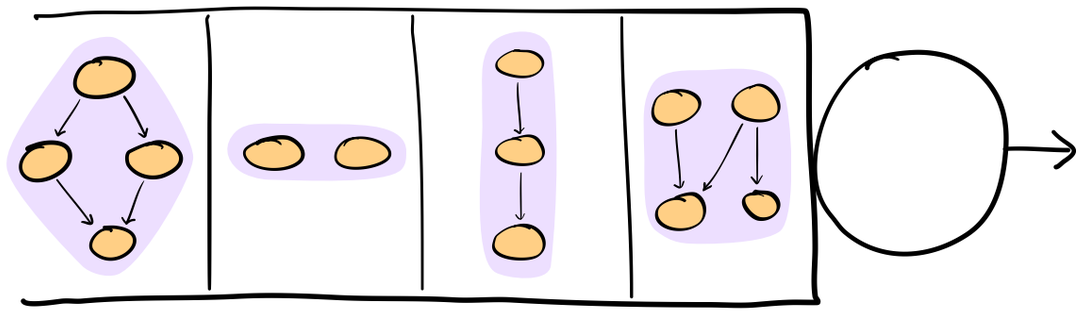


Prior Work: Single-Task

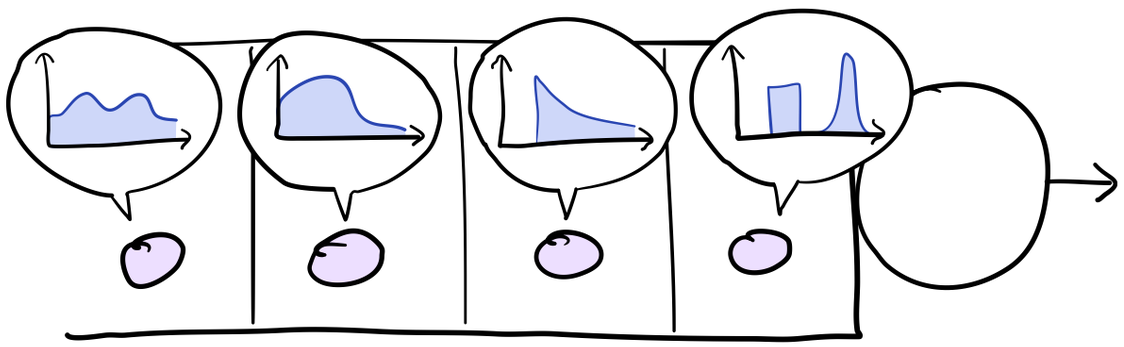


Gittins index policy optimal

New Problem: Multitask



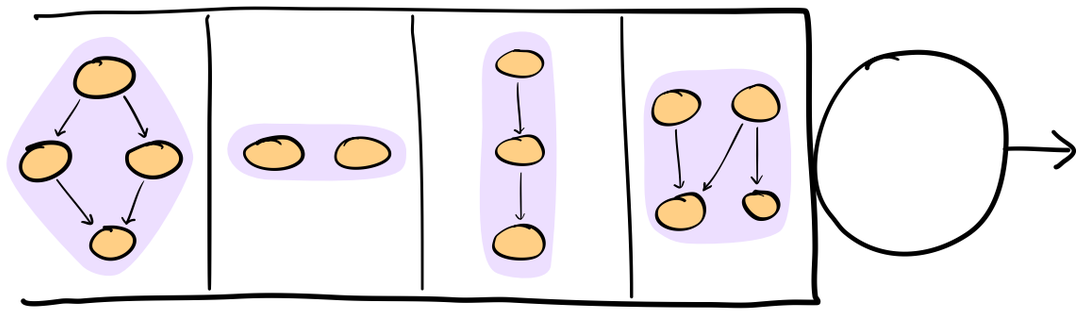
Prior Work: Single-Task



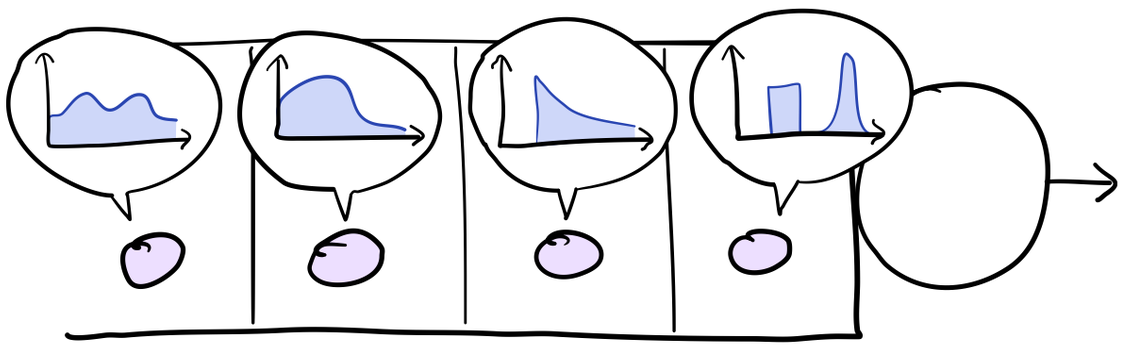
Gittins index policy optimal

Key property: analyzes each job *independently*

New Problem: Multitask



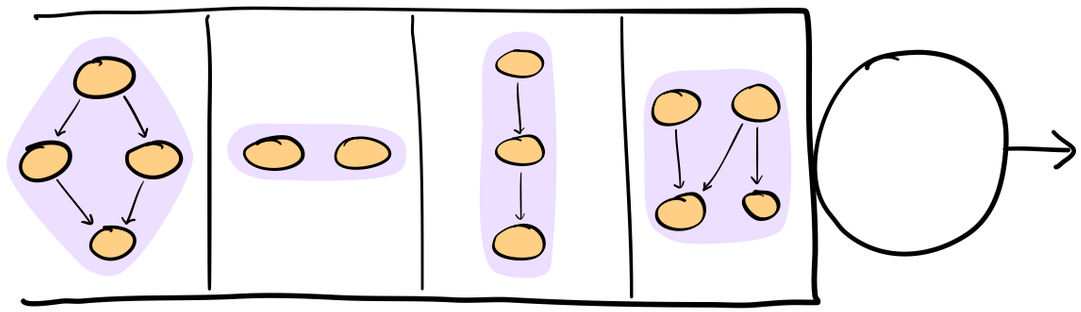
Prior Work: Single-Task



Gittins index policy optimal

Key property: analyzes each job *independently*

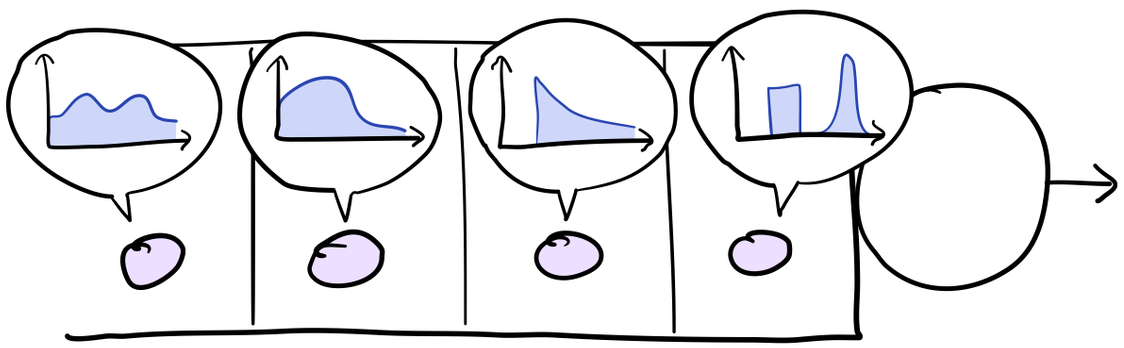
New Problem: Multitask



Goal:

- Which **job**?
- Which **task**?

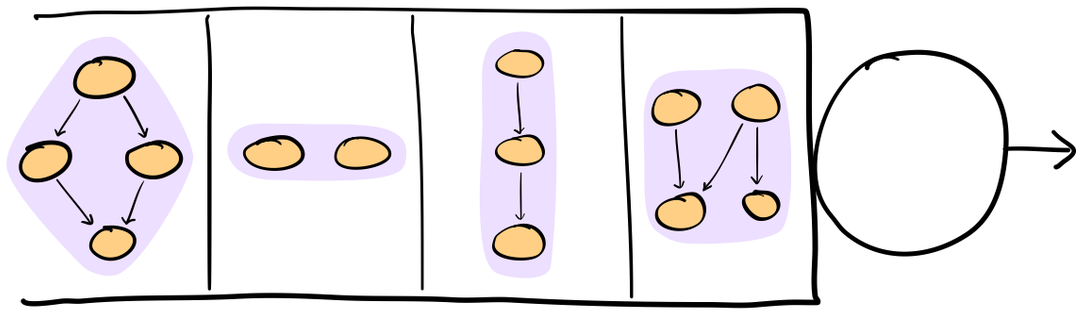
Prior Work: Single-Task



Gittins index policy optimal

Key property: analyzes each job *independently*

New Problem: Multitask



Goal:

- Which **job**?
- Which **task**?

Gittins index??

Gittins index

Gittins index

many equivalent definitions

- Efficiency function [Gittins ’79]
- Fair charge [Weber ‘92]
- Retirement option [Whittle ‘80]
- Restart-in-state [Katehakis and Veinott ‘87]

Gittins index

many equivalent definitions

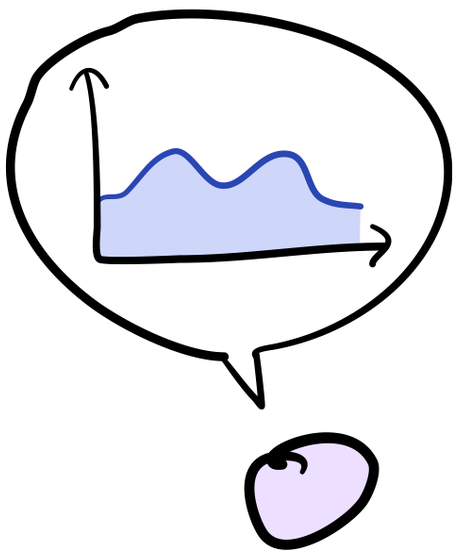
- Efficiency function [Gittins ’79]
- Fair charge [Weber ‘92]
- Retirement option [Whittle ‘80]
- Restart-in-state [Katehakis and Veinott ‘87]
- **Single-job profit** (this talk)

Gittins index

many equivalent definitions

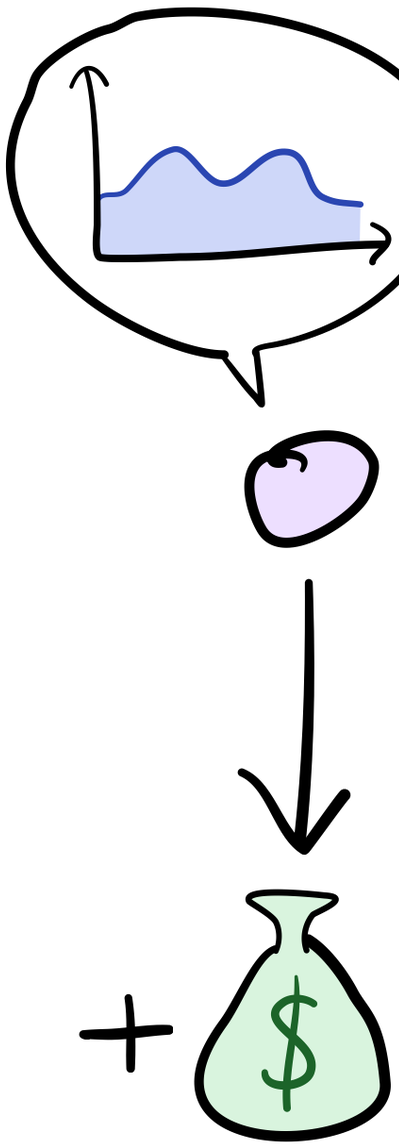
- Efficiency function [Gittins ’79]
- Fair charge [Weber ‘92]
- Retirement option [Whittle ‘80]
- Restart-in-state [Katehakis and Veinott ‘87]
- **Single-job profit** (this talk)
  - Natural definition for multitask jobs

Single-Job Profit



Game with a **job** and potential **reward**

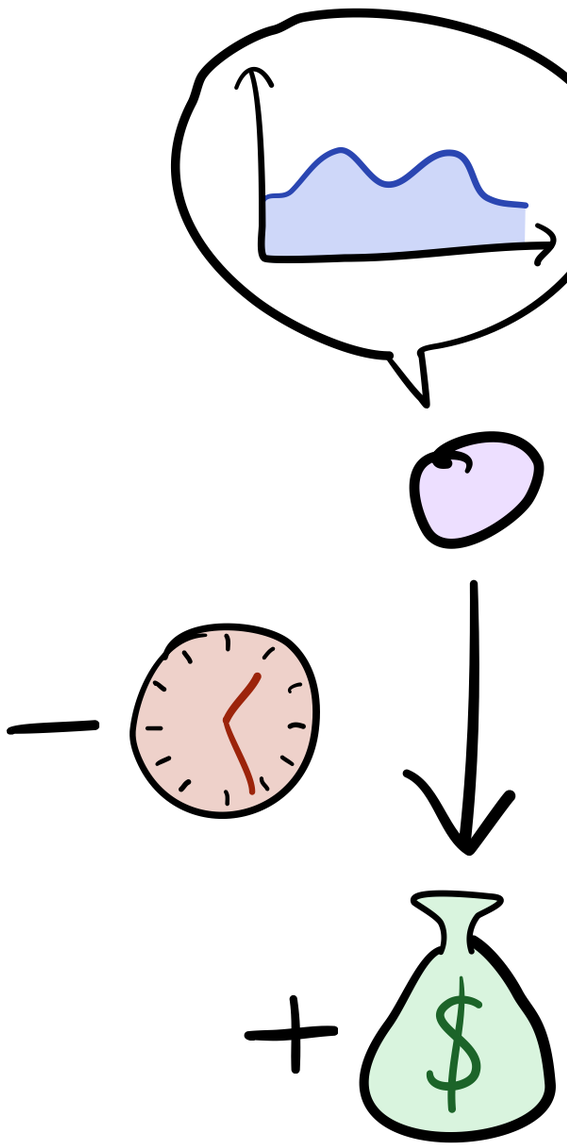
Single-Job Profit



Game with a **job** and potential **reward**

Run **job** as long as we like, get **reward** if we complete it

Single-Job Profit

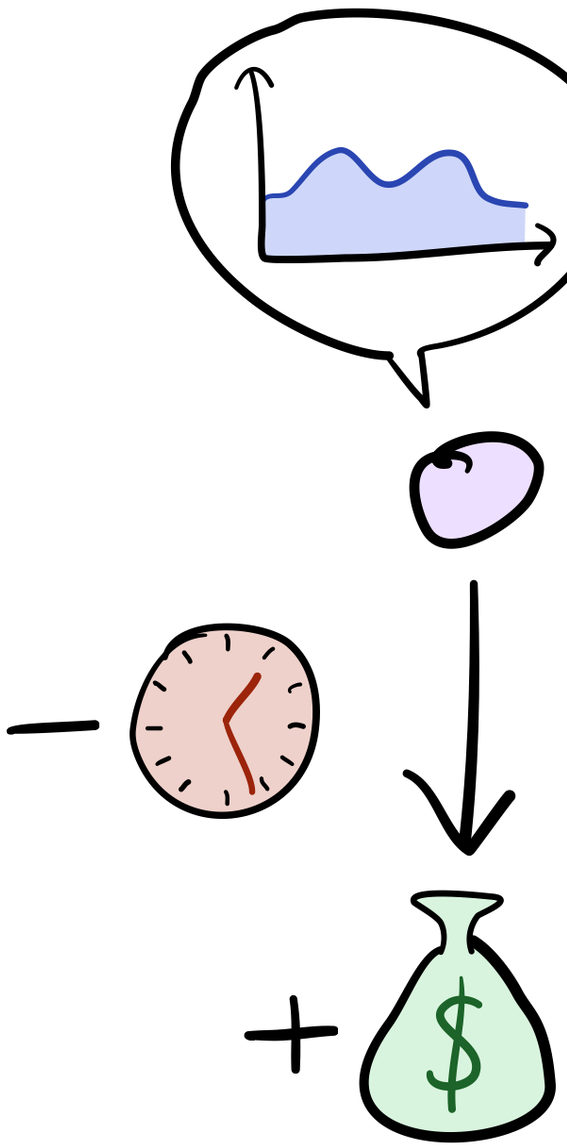


Game with a **job** and potential **reward**

Run **job** as long as we like, get **reward** if we complete it

Pay for **time** spent running job

Single-Job Profit



Game with a **job** and potential **reward**

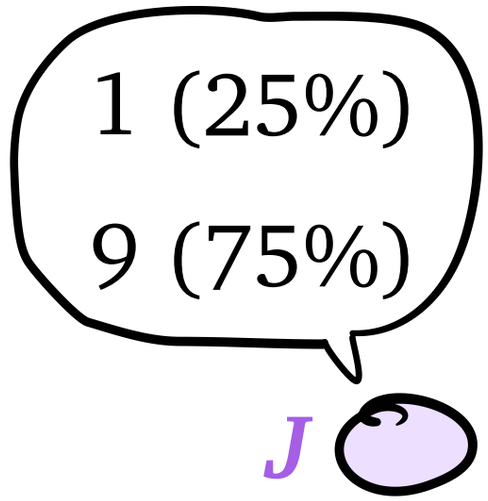
Run **job** as long as we like, get **reward** if we complete it

Pay for **time** spent running job

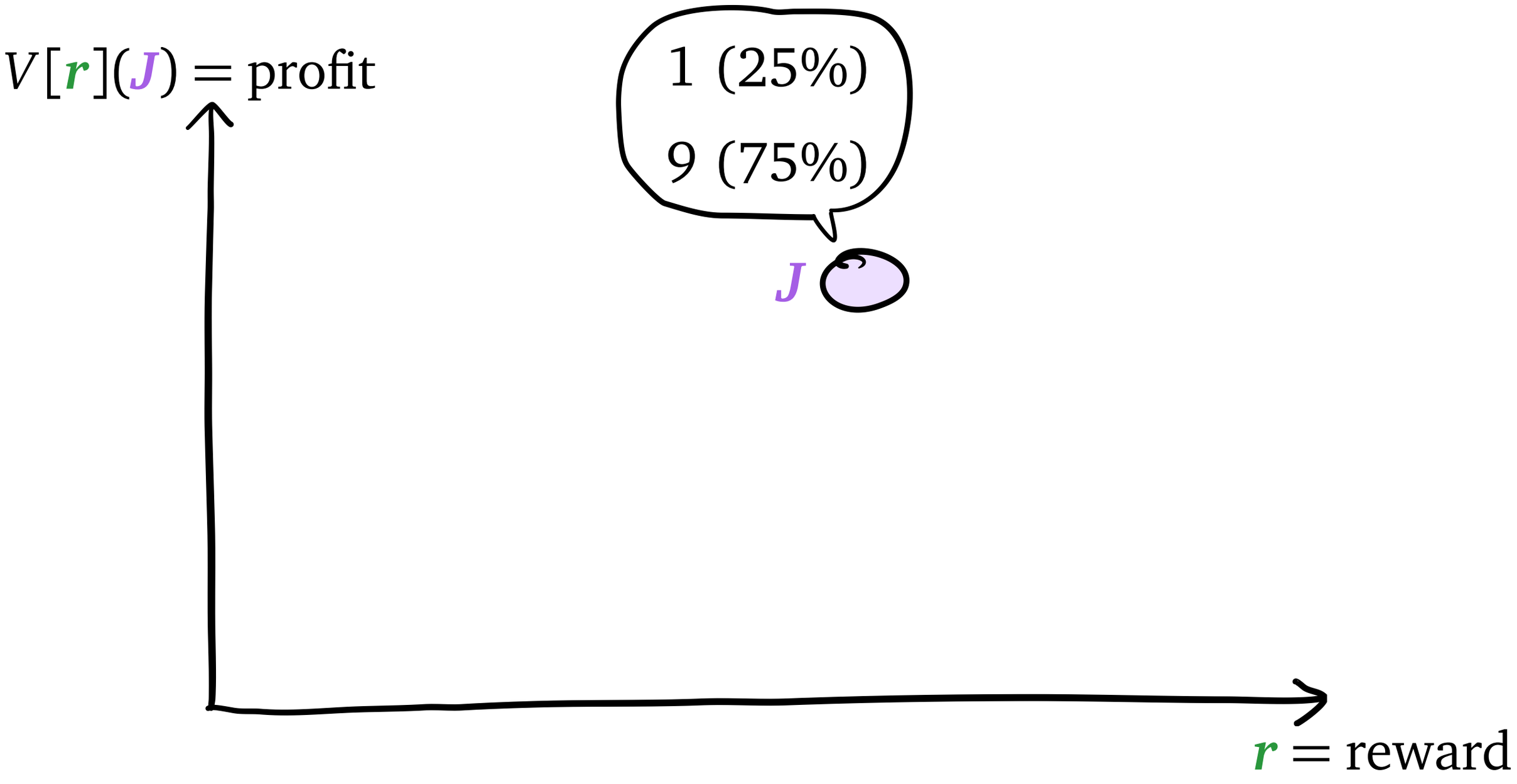
Goal: maximize *profit*,

E[**reward** recieved − **time** spent]

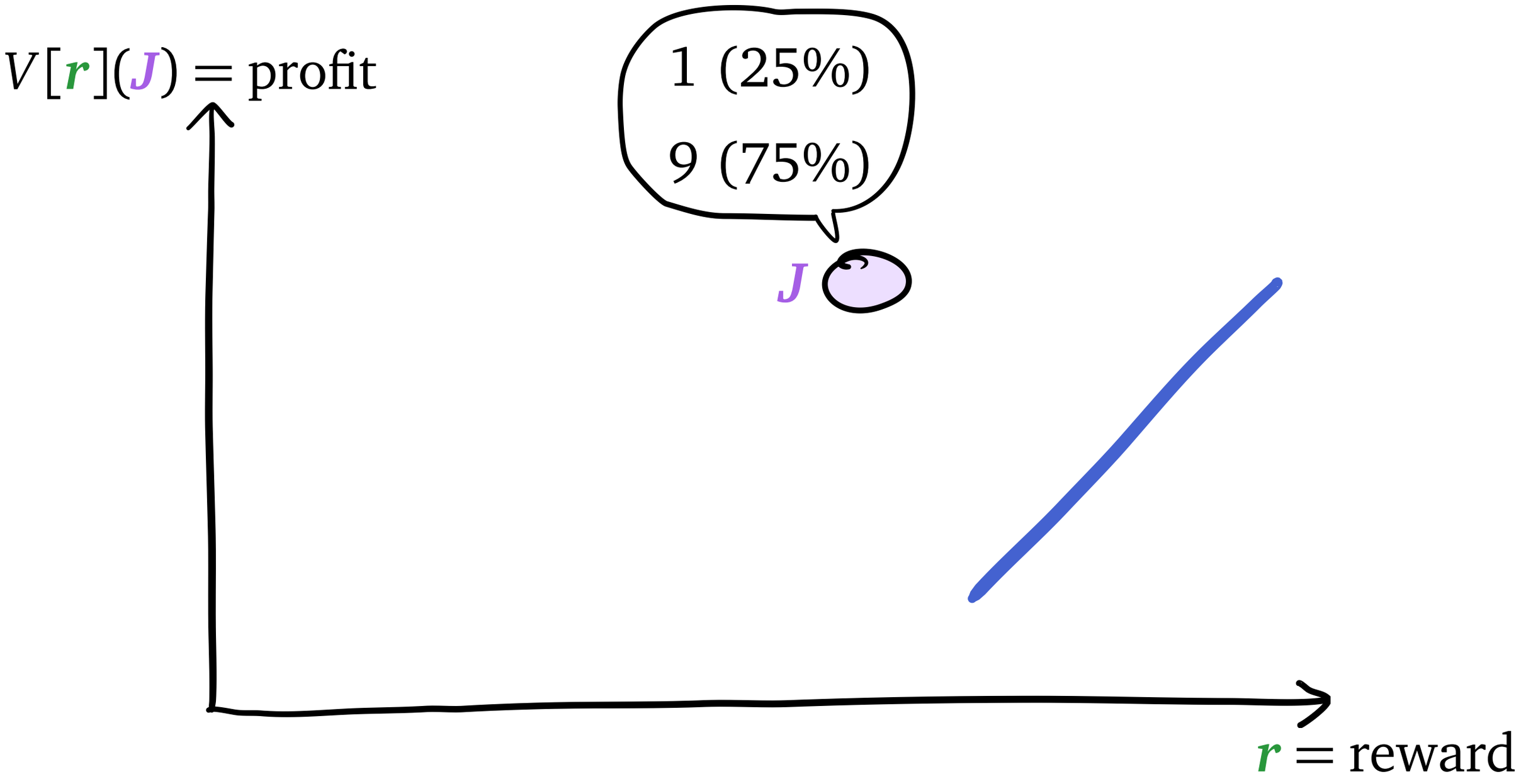
Example



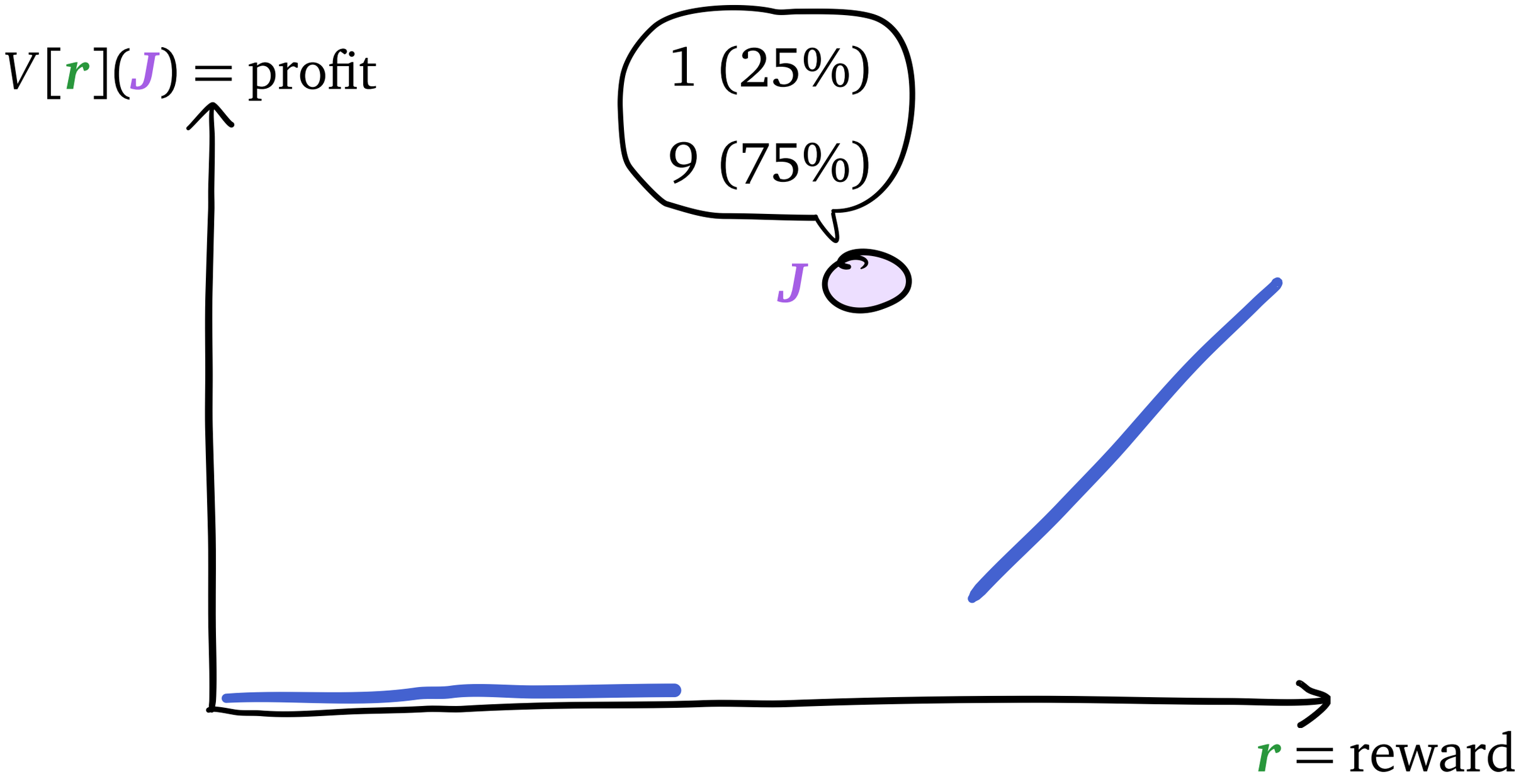
Example



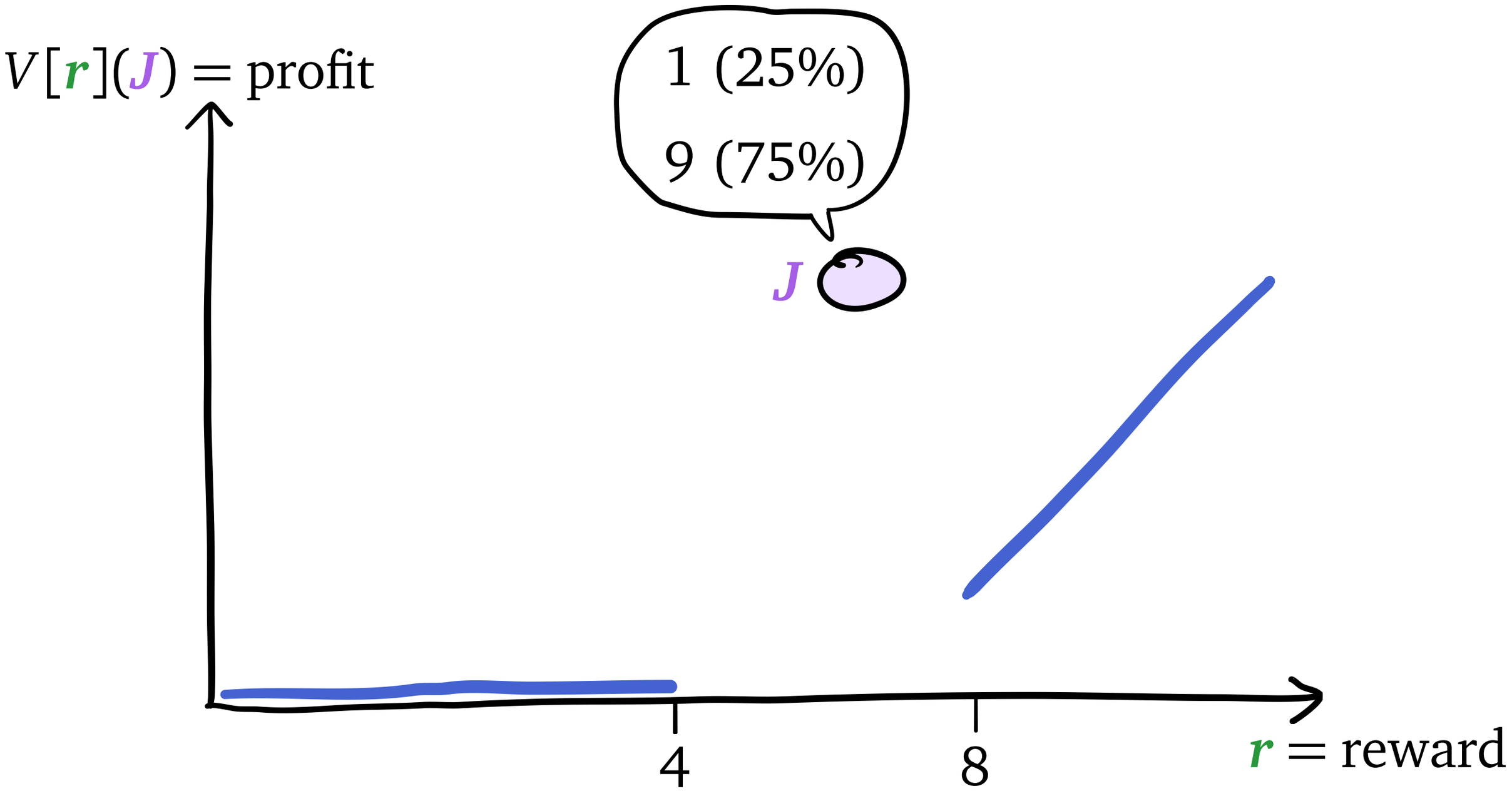
Example



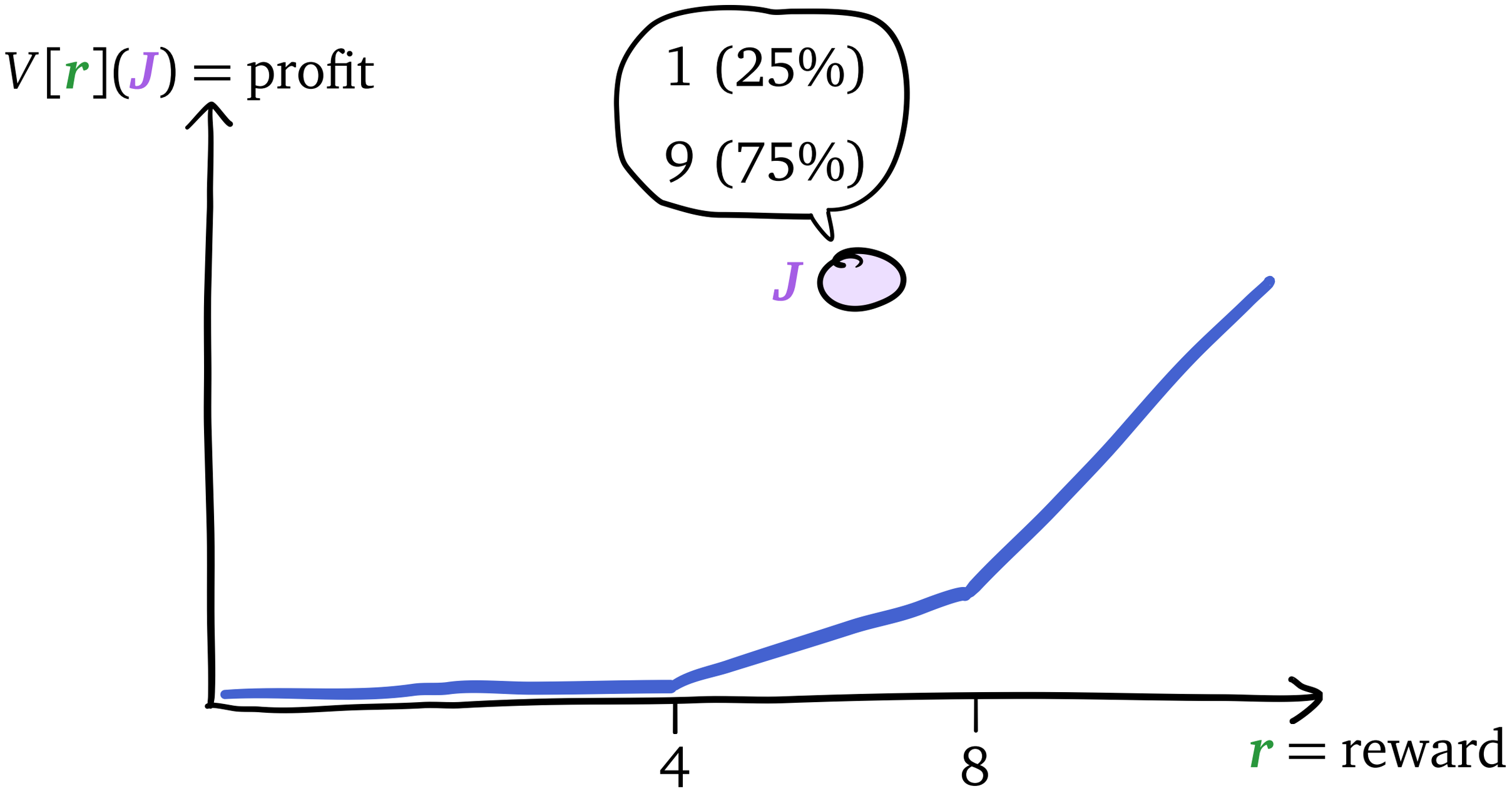
Example



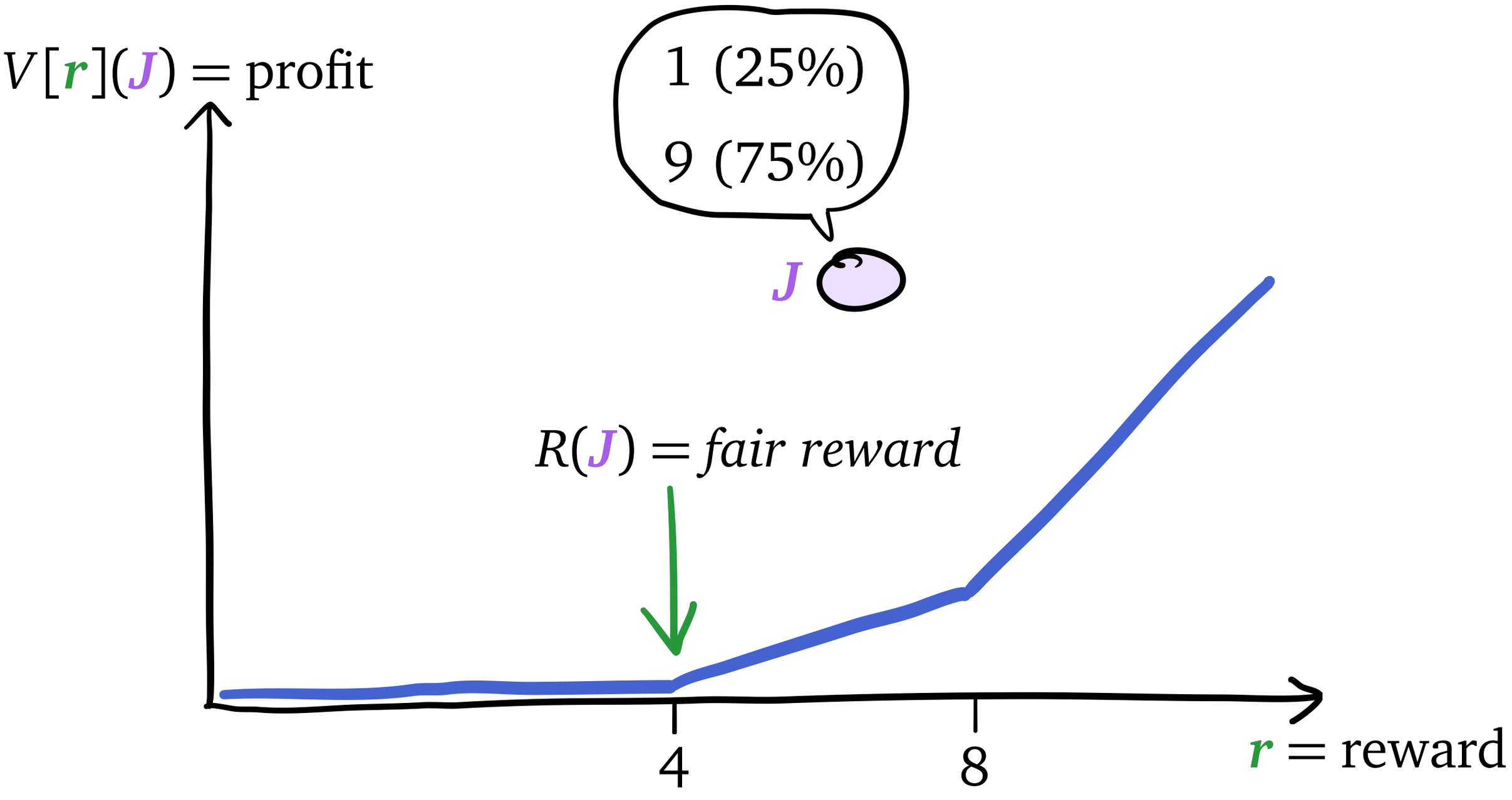
Example



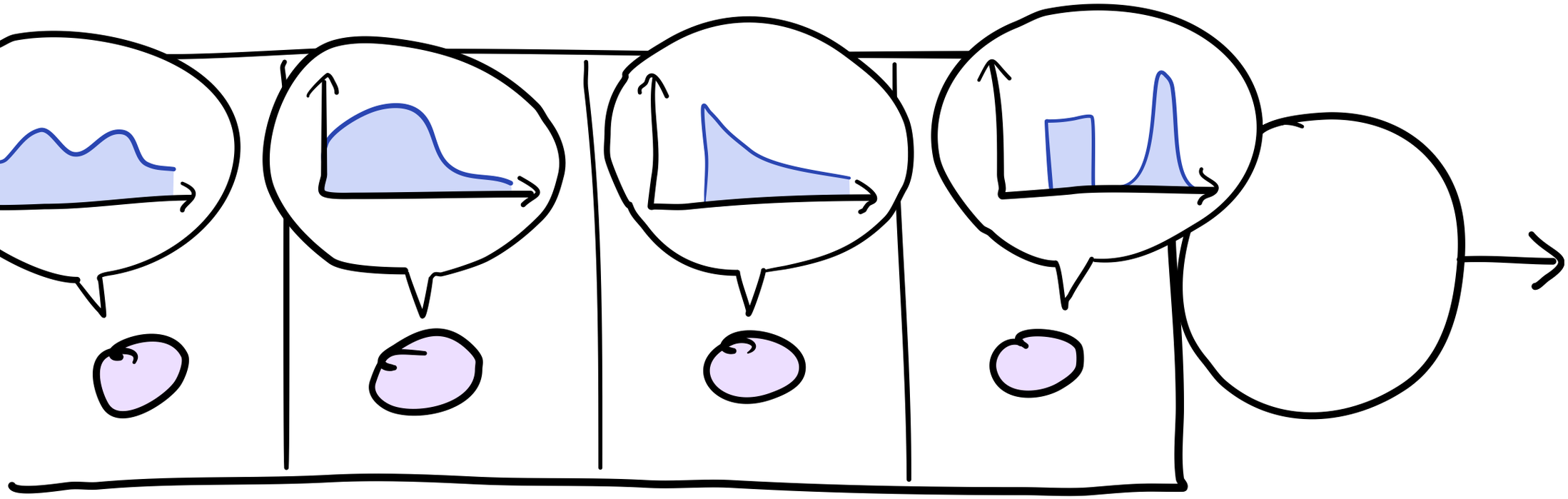
Example



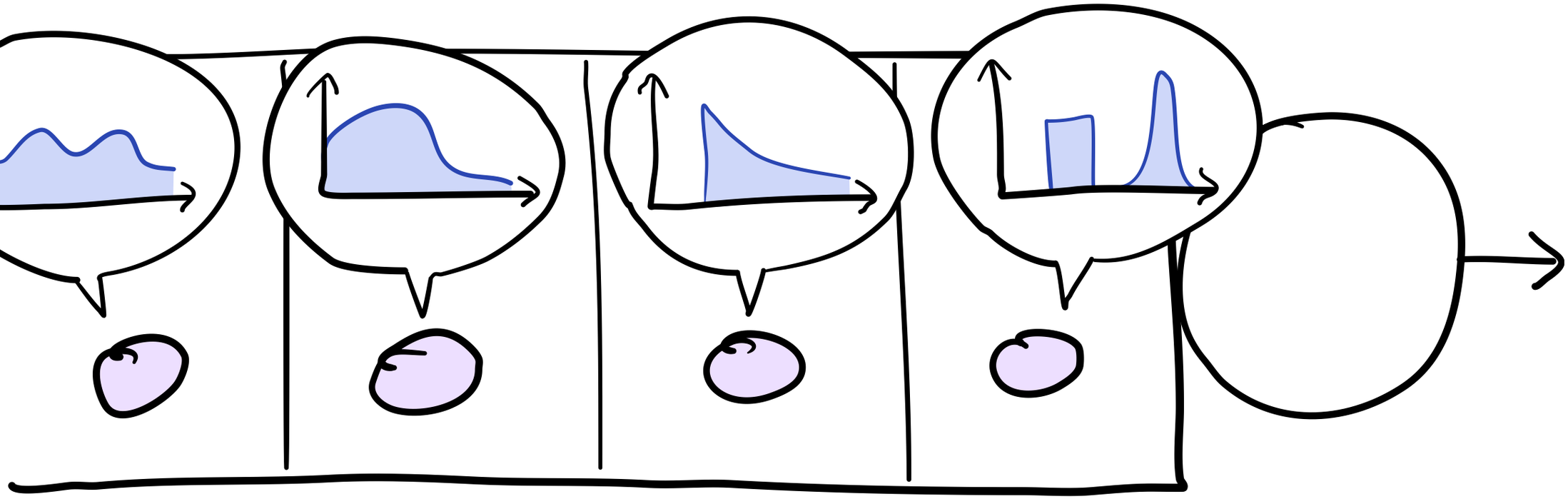
Example



Fair Reward in Scheduling

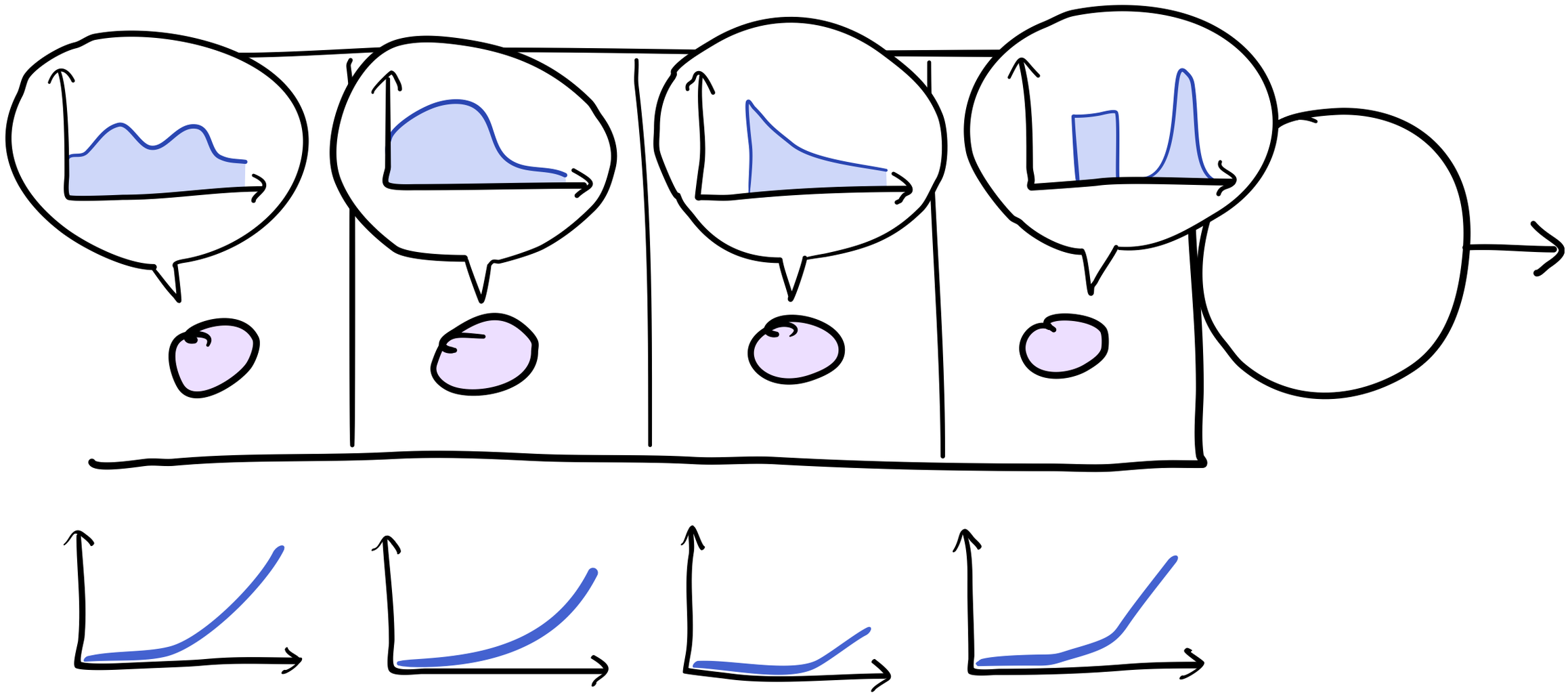


Fair Reward in Scheduling



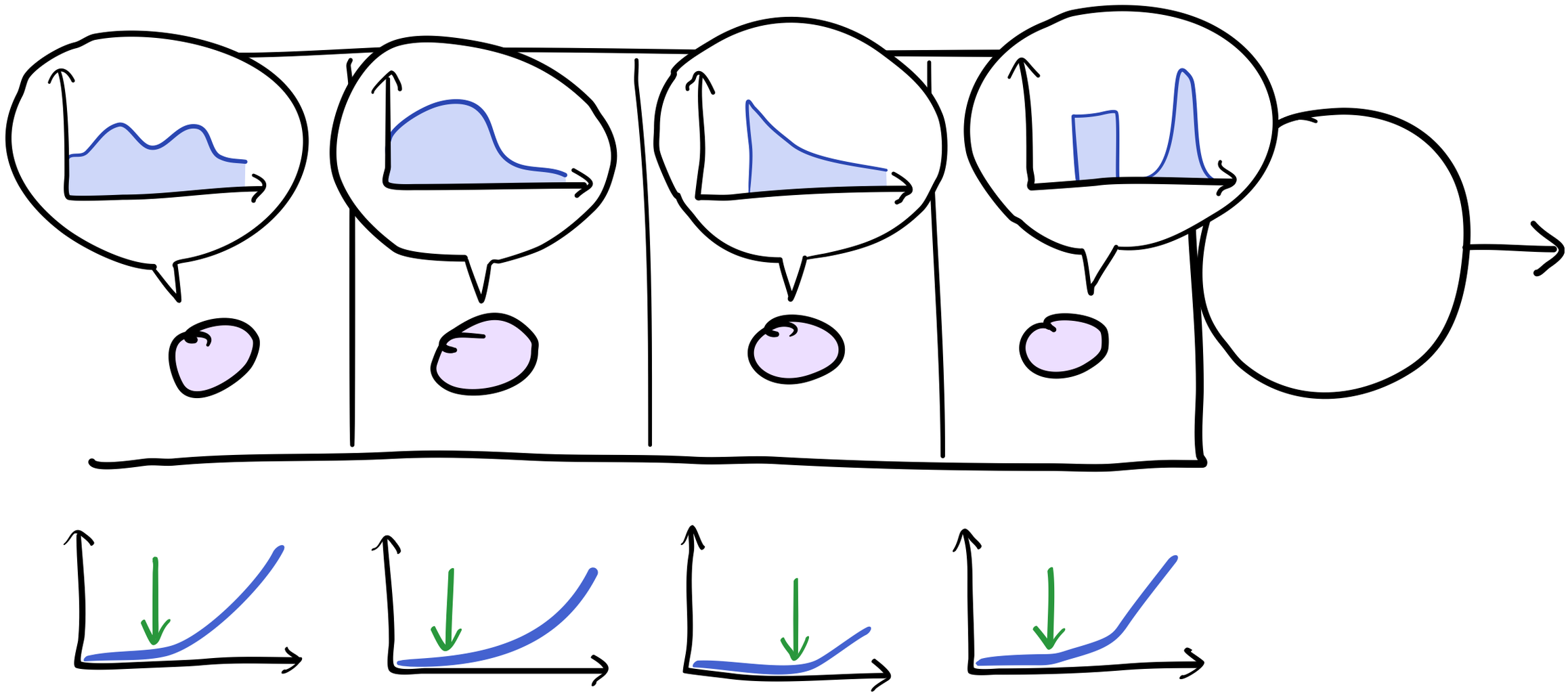
Theorem: optimal to serve
job of *minimum fair reward*

Fair Reward in Scheduling



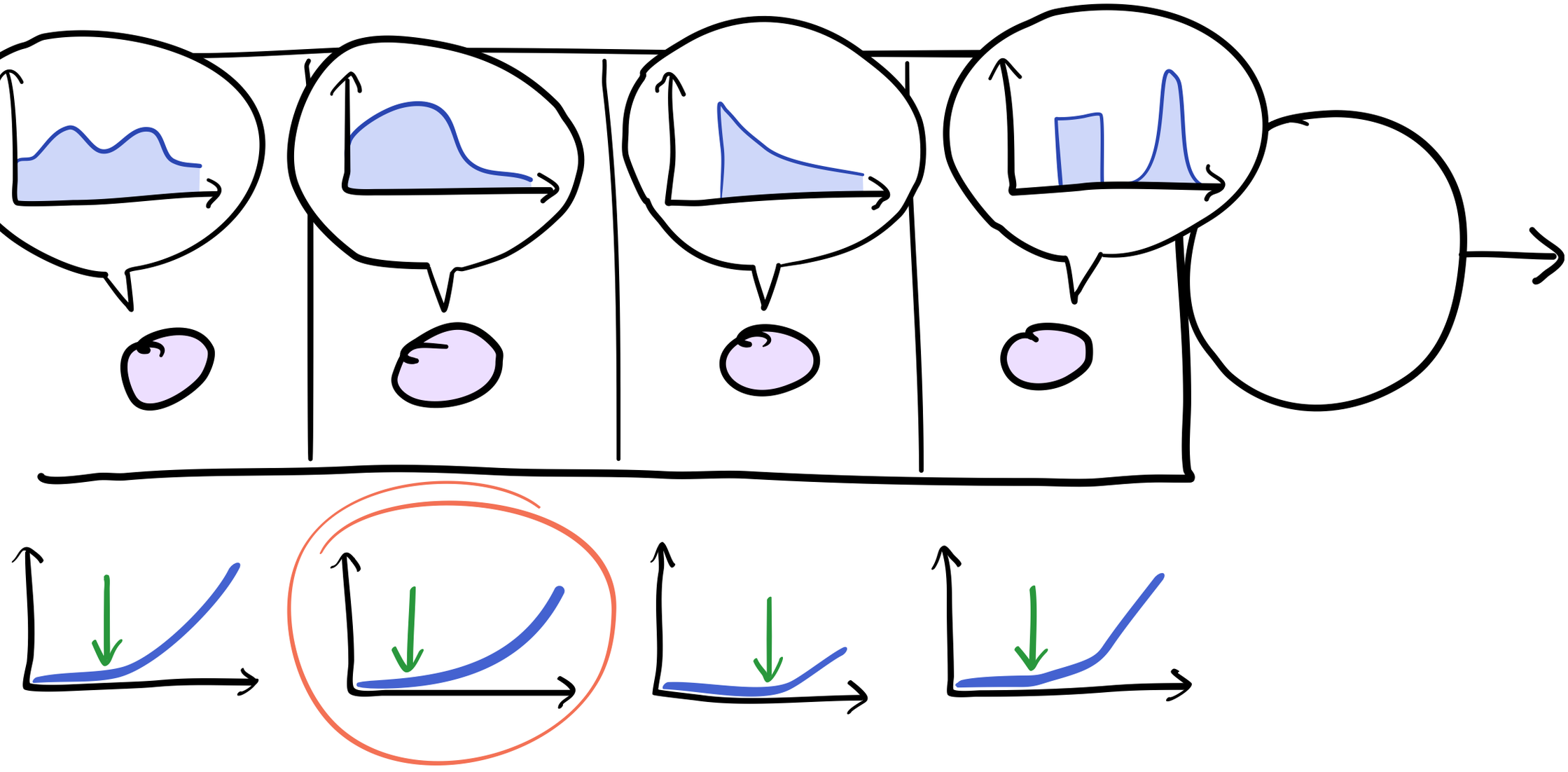
Theorem: optimal to serve job of *minimum fair reward*

Fair Reward in Scheduling



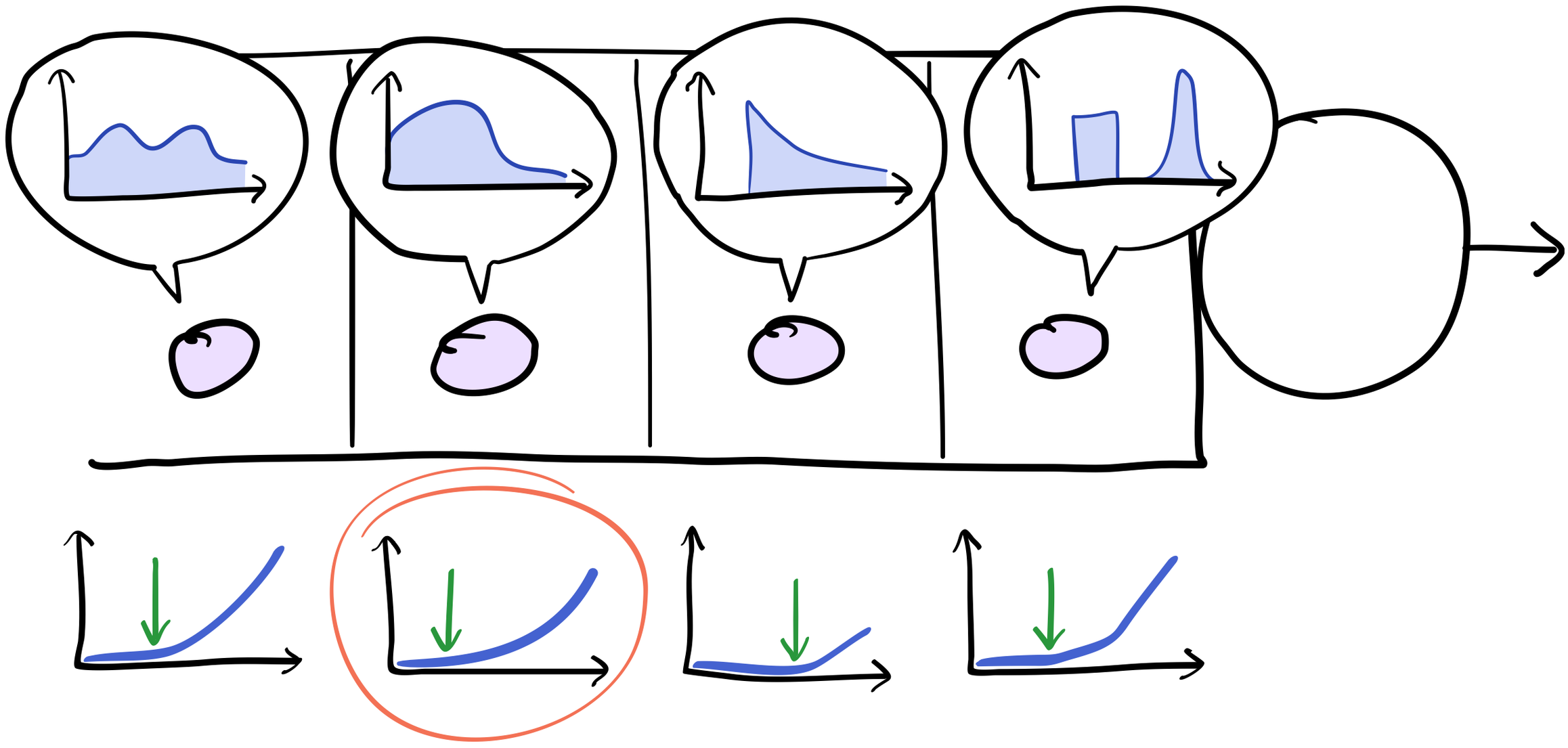
Theorem: optimal to serve job of *minimum fair reward*

Fair Reward in Scheduling



Theorem: optimal to serve job of *minimum fair reward*

Fair Reward in Scheduling

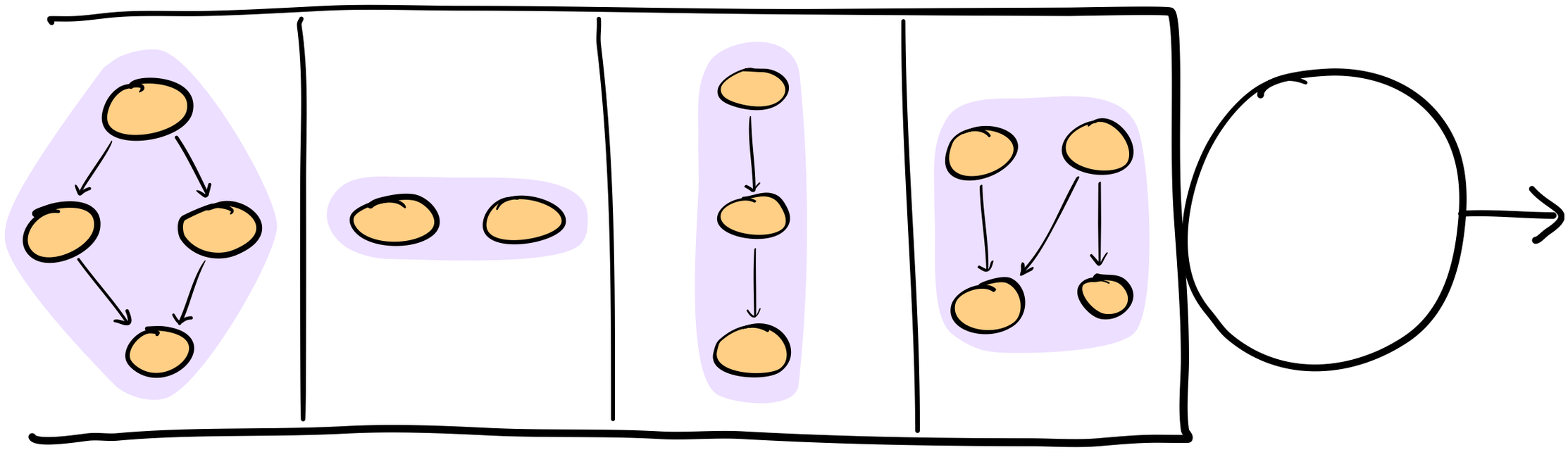


Theorem: optimal to serve job of *minimum fair reward*

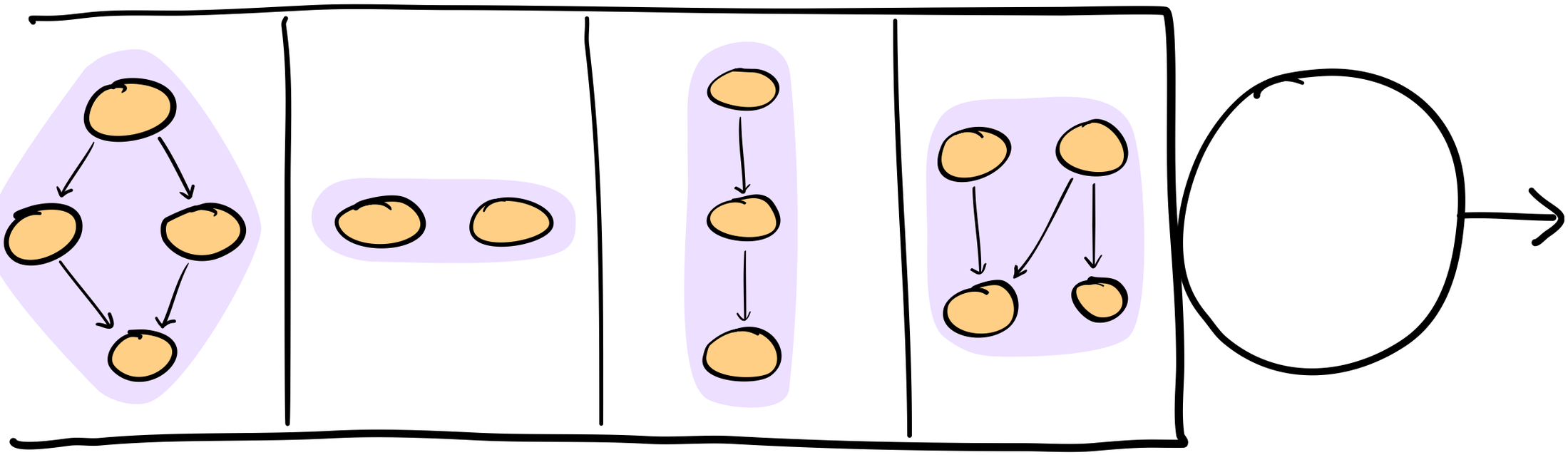
$$G(J) = \frac{1}{R(J)}$$

Gittins index

Generalizing to Multitask

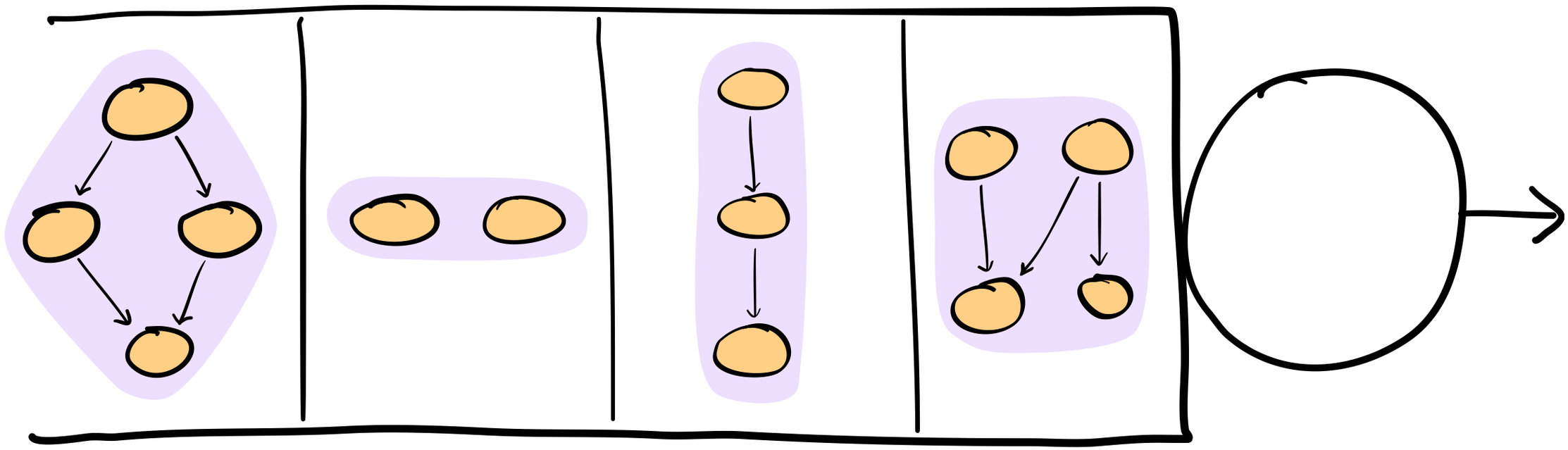


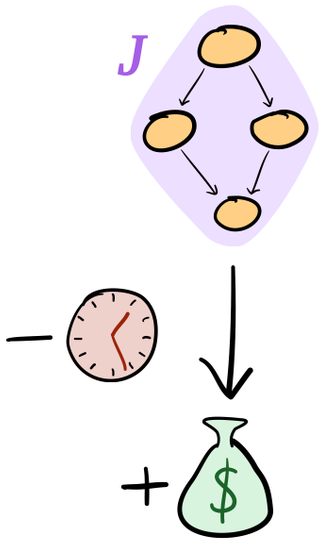
Generalizing to Multitask



Single-job profit easily generalizes

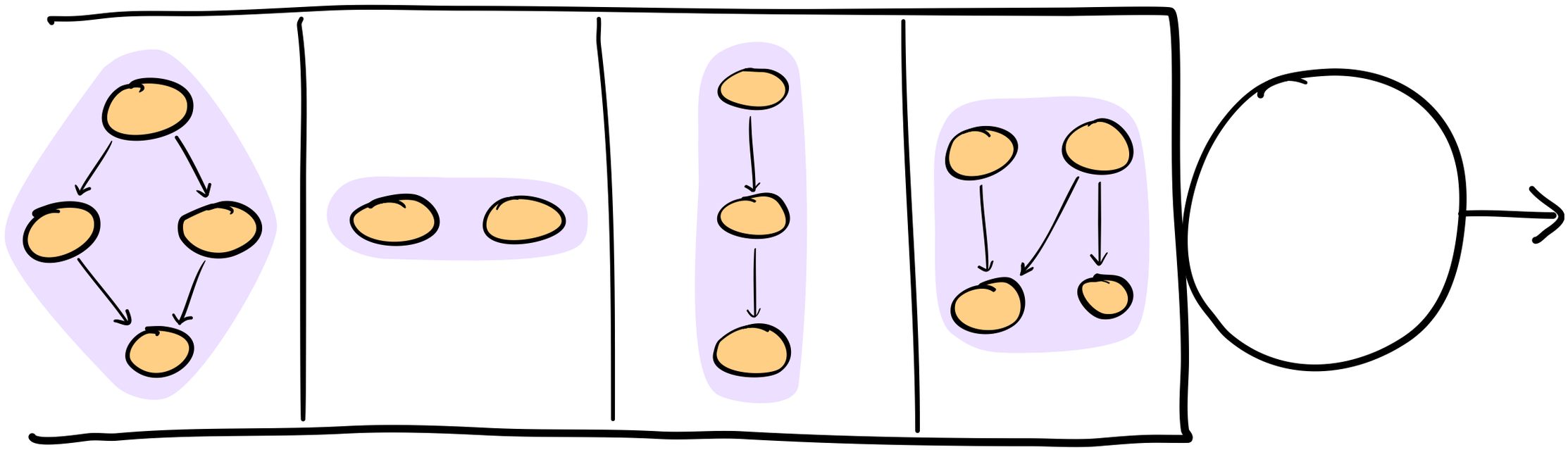
Generalizing to Multitask

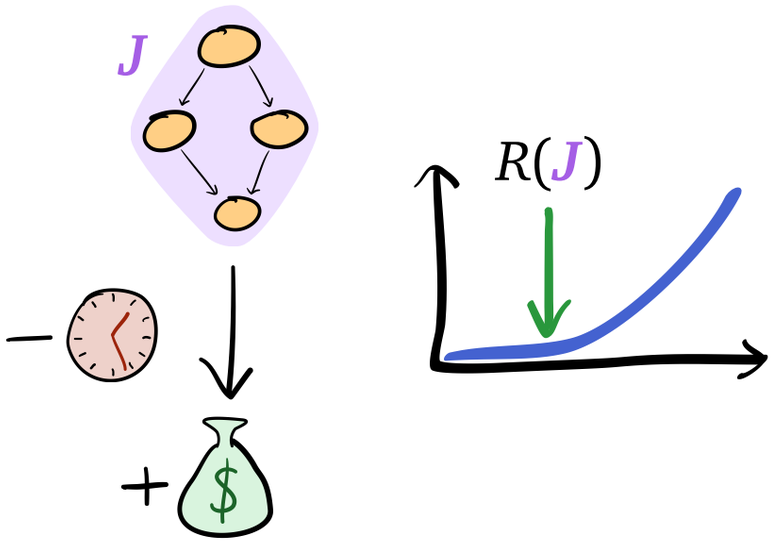




Single-job profit easily generalizes

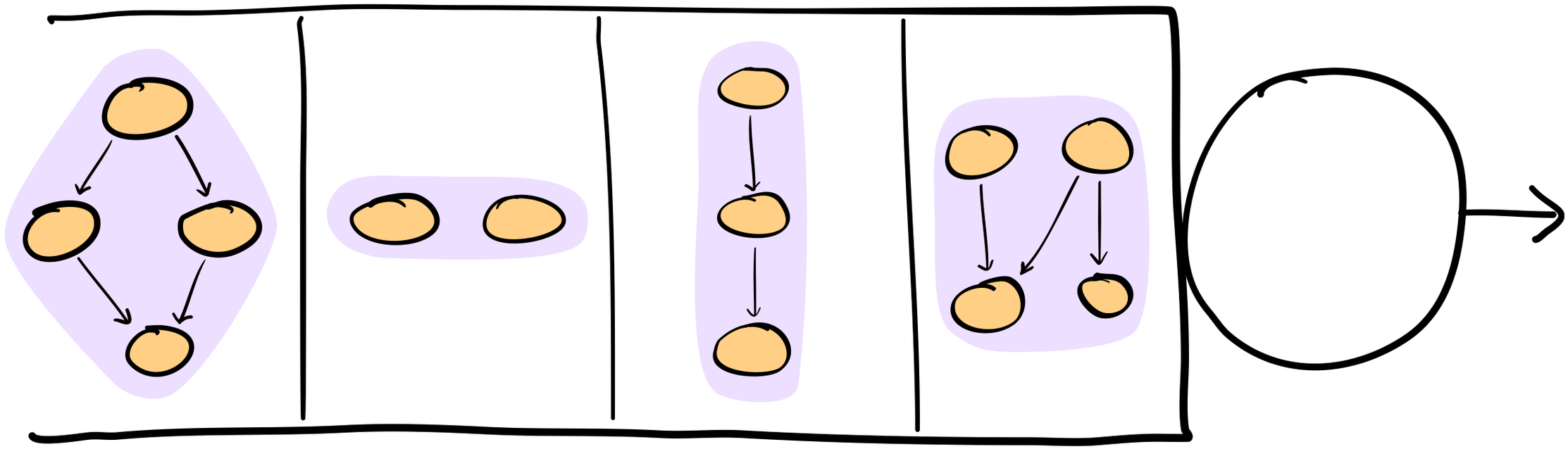
Generalizing to Multitask

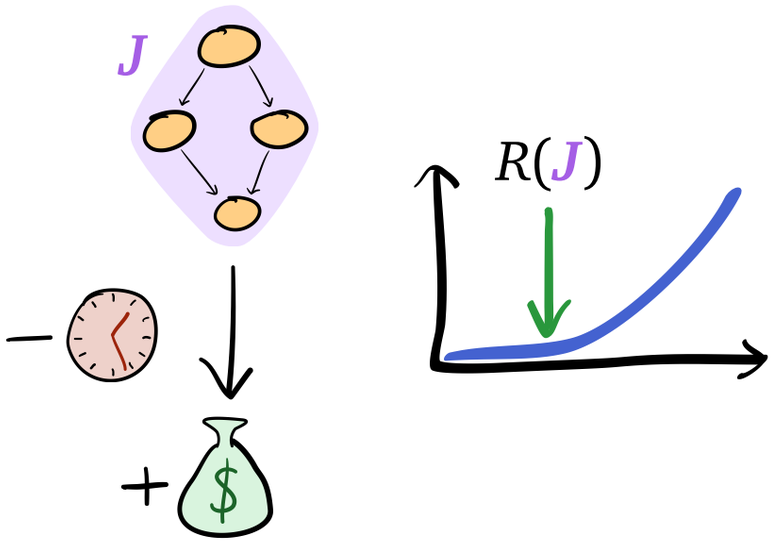




Single-job profit easily generalizes

Generalizing to Multitask

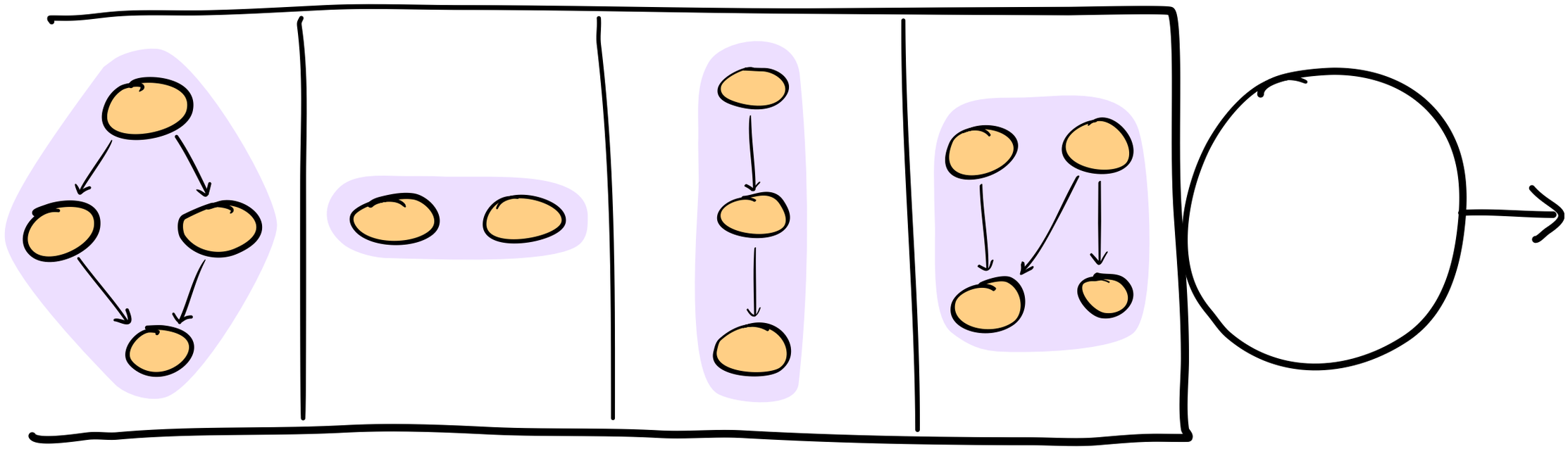


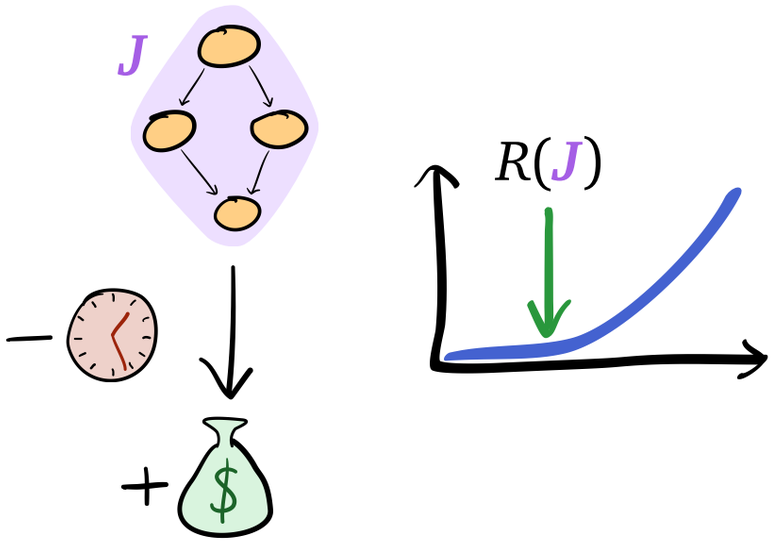


Single-job profit easily generalizes

⚠ **Obstacles**:

Generalizing to Multitask



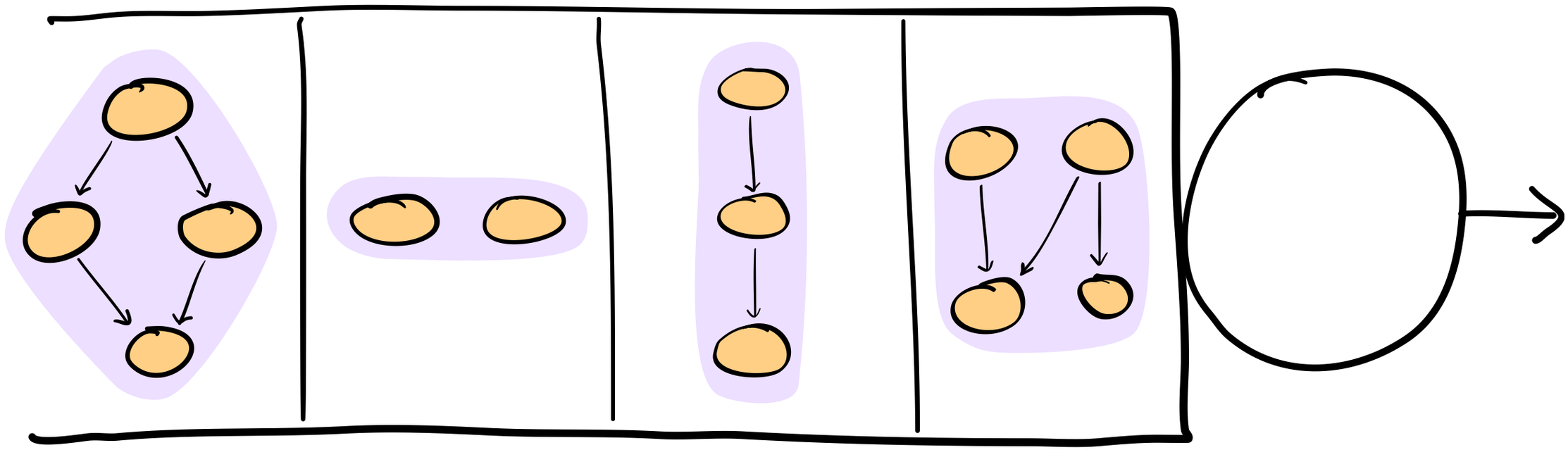


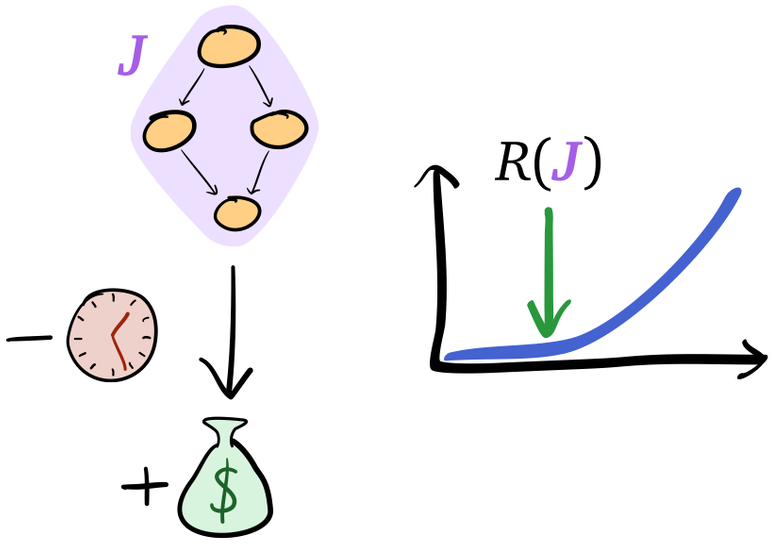
Single-job profit easily generalizes

Obstacles:

- Hard to compute profit

Generalizing to Multitask



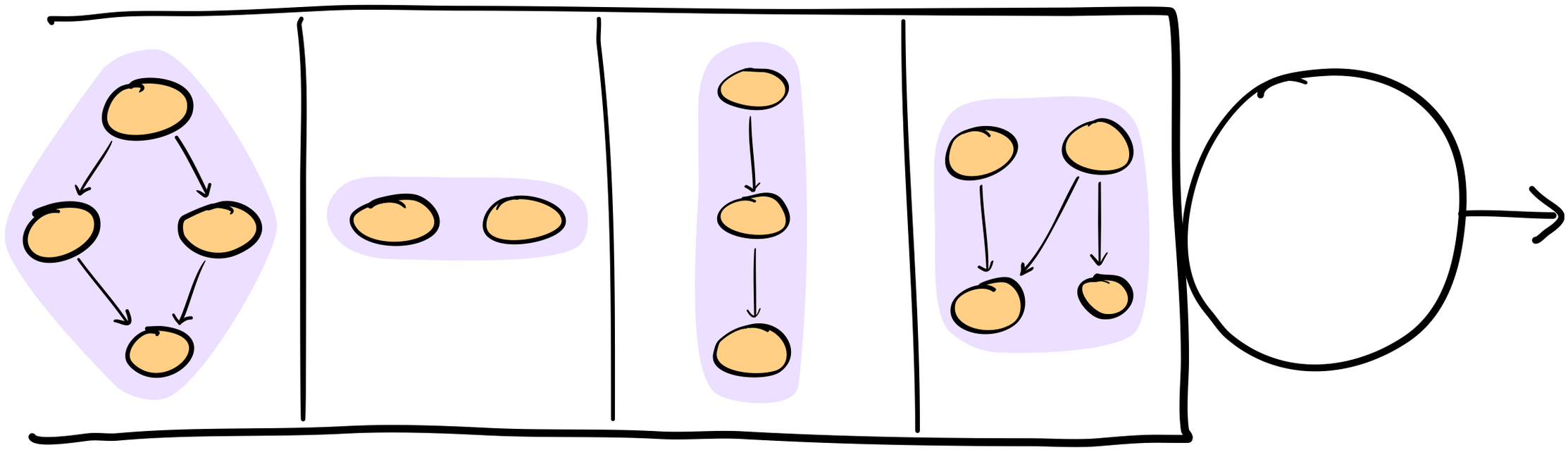


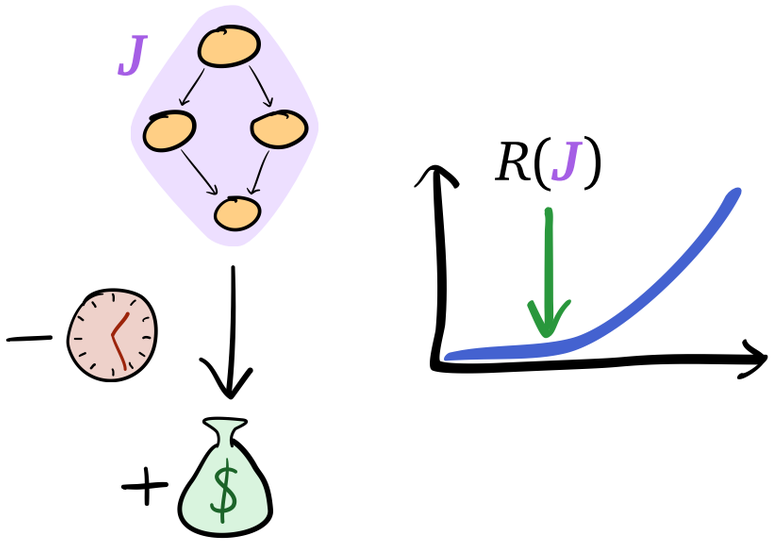
Single-job profit easily generalizes

Obstacles:

- Hard to compute profit
- Fair reward policy not always optimal

Generalizing to Multitask





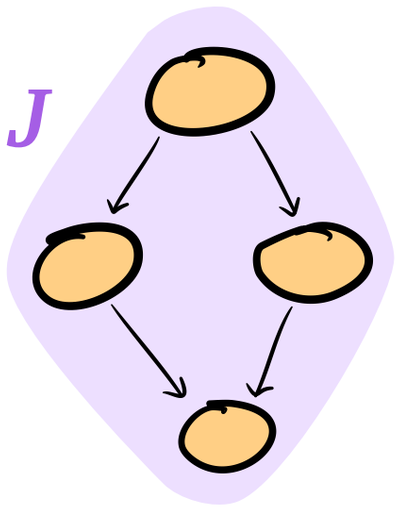
Single-job profit easily generalizes

Obstacles:

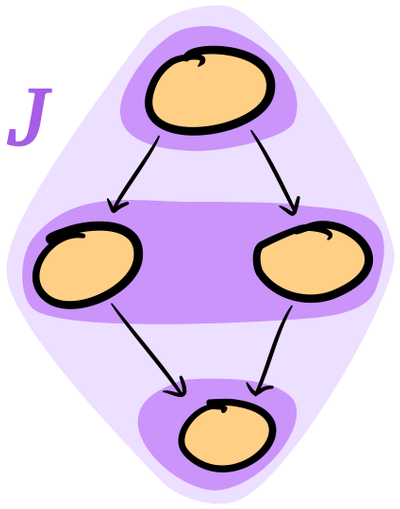
this talk

- Hard to compute profit
- Fair reward policy not always optimal

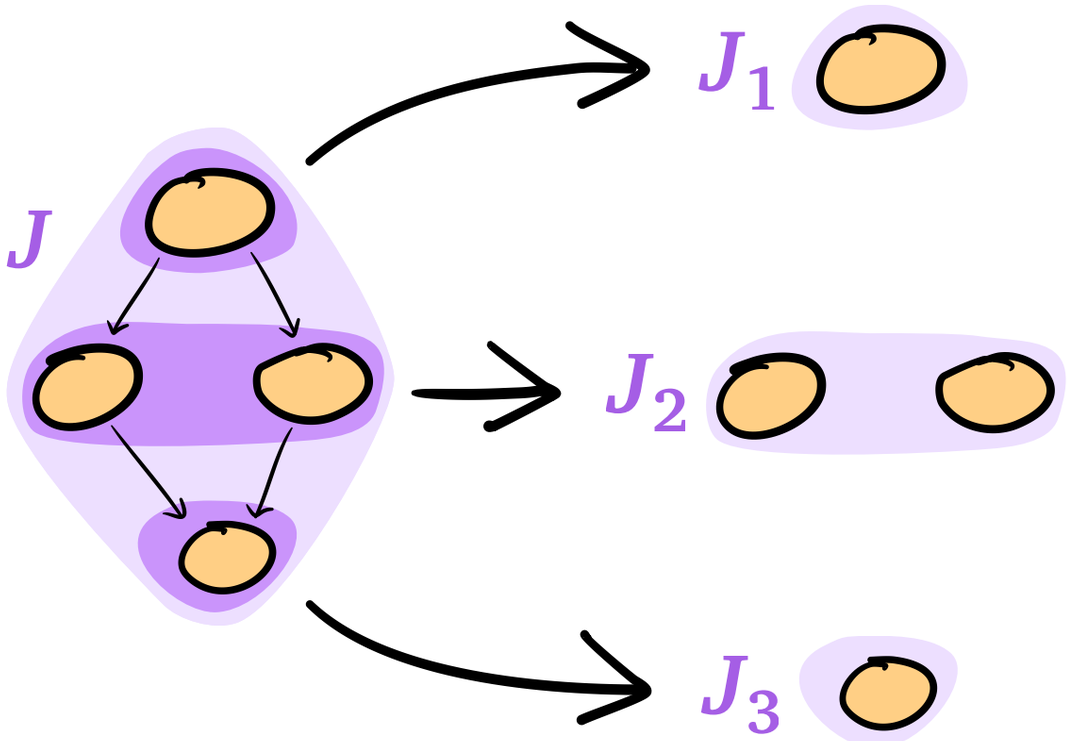
Splitting Big Jobs



Splitting Big Jobs

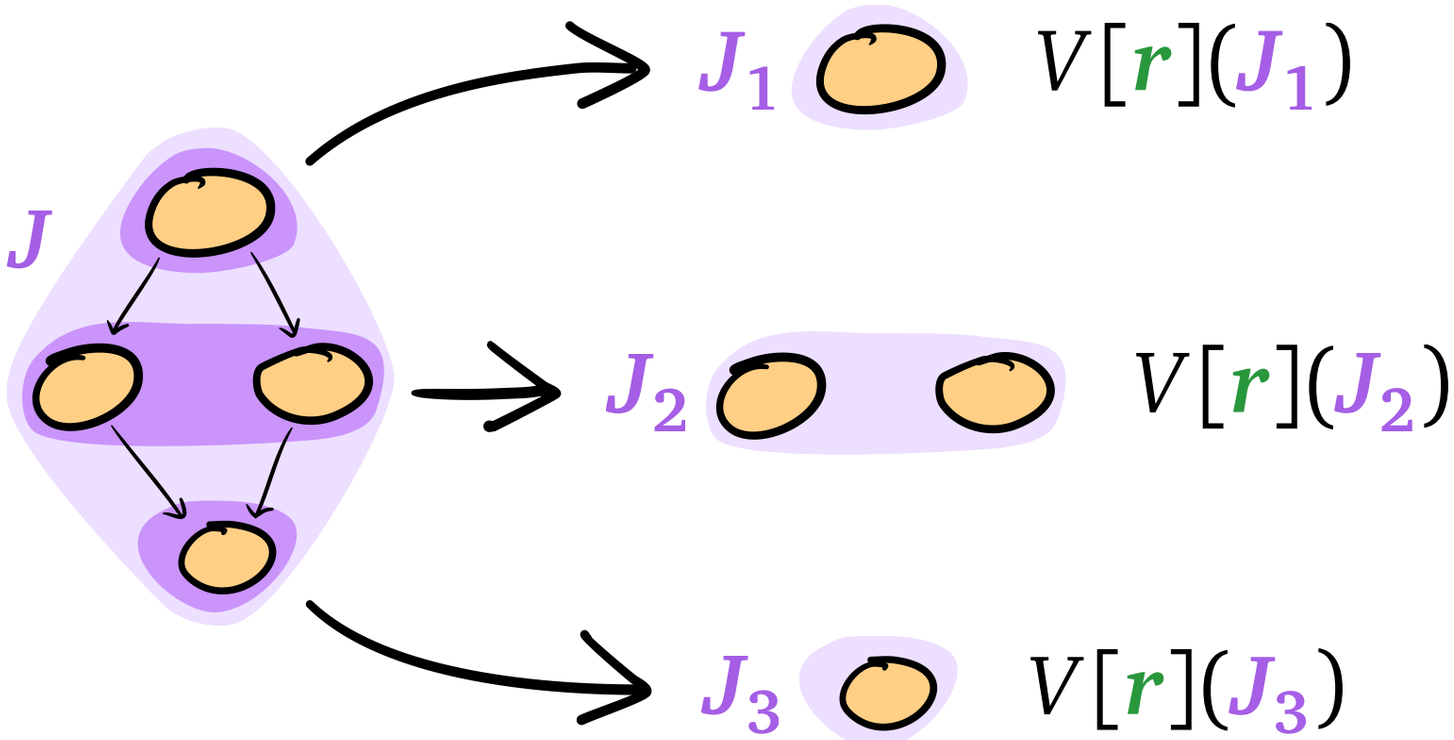


Splitting Big Jobs



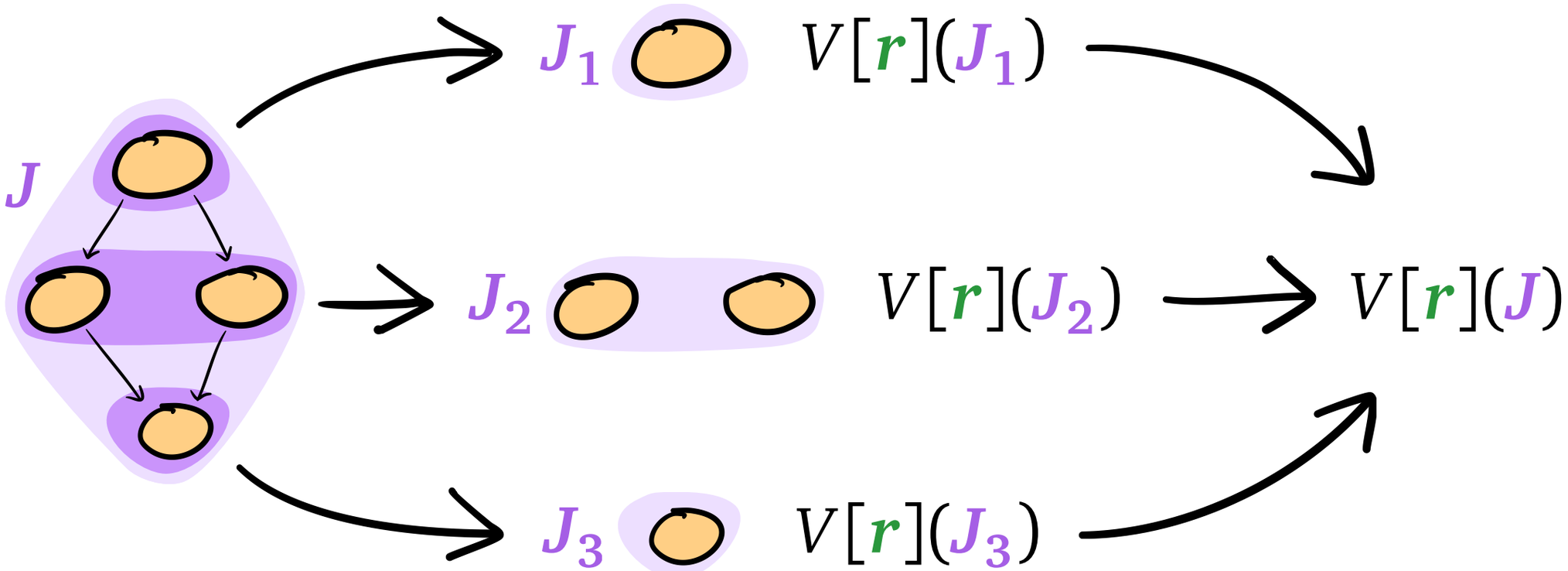
phases

Splitting Big Jobs

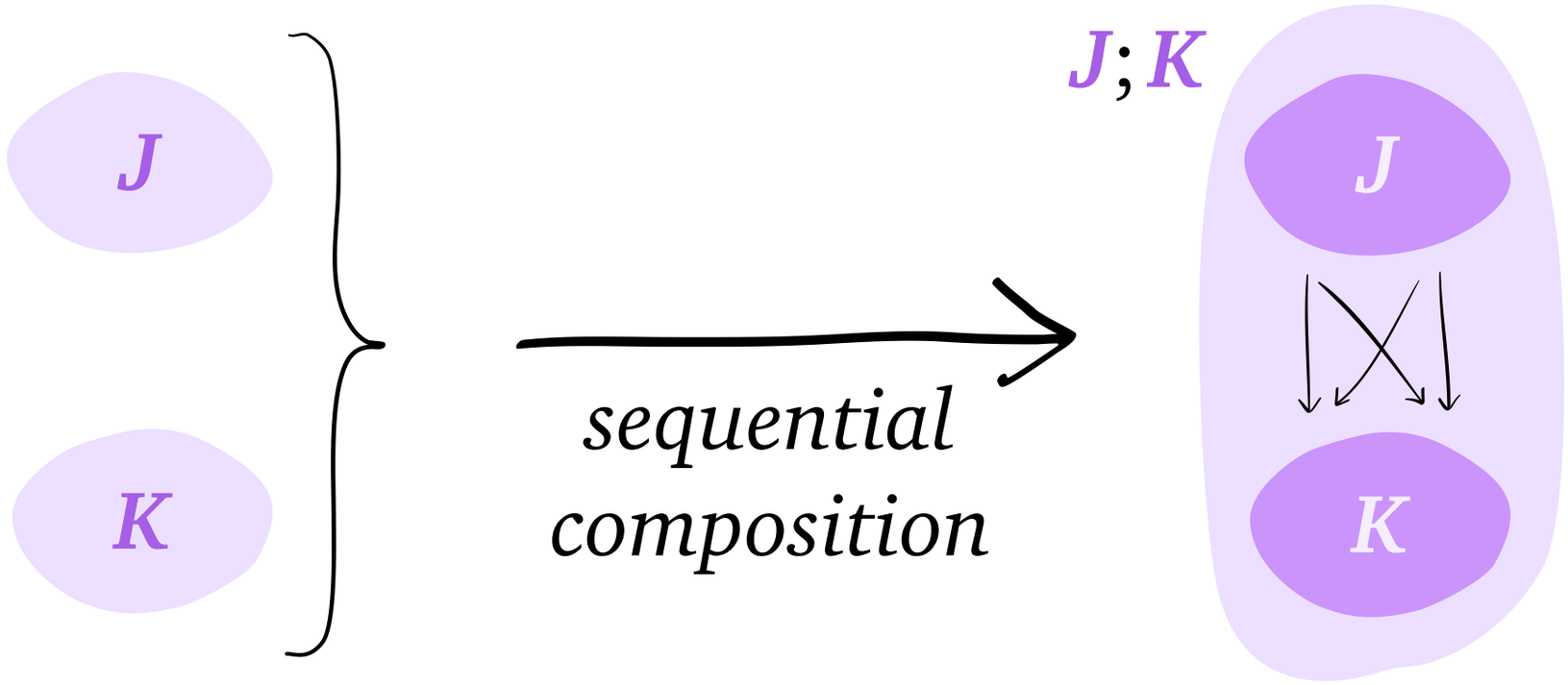


phases

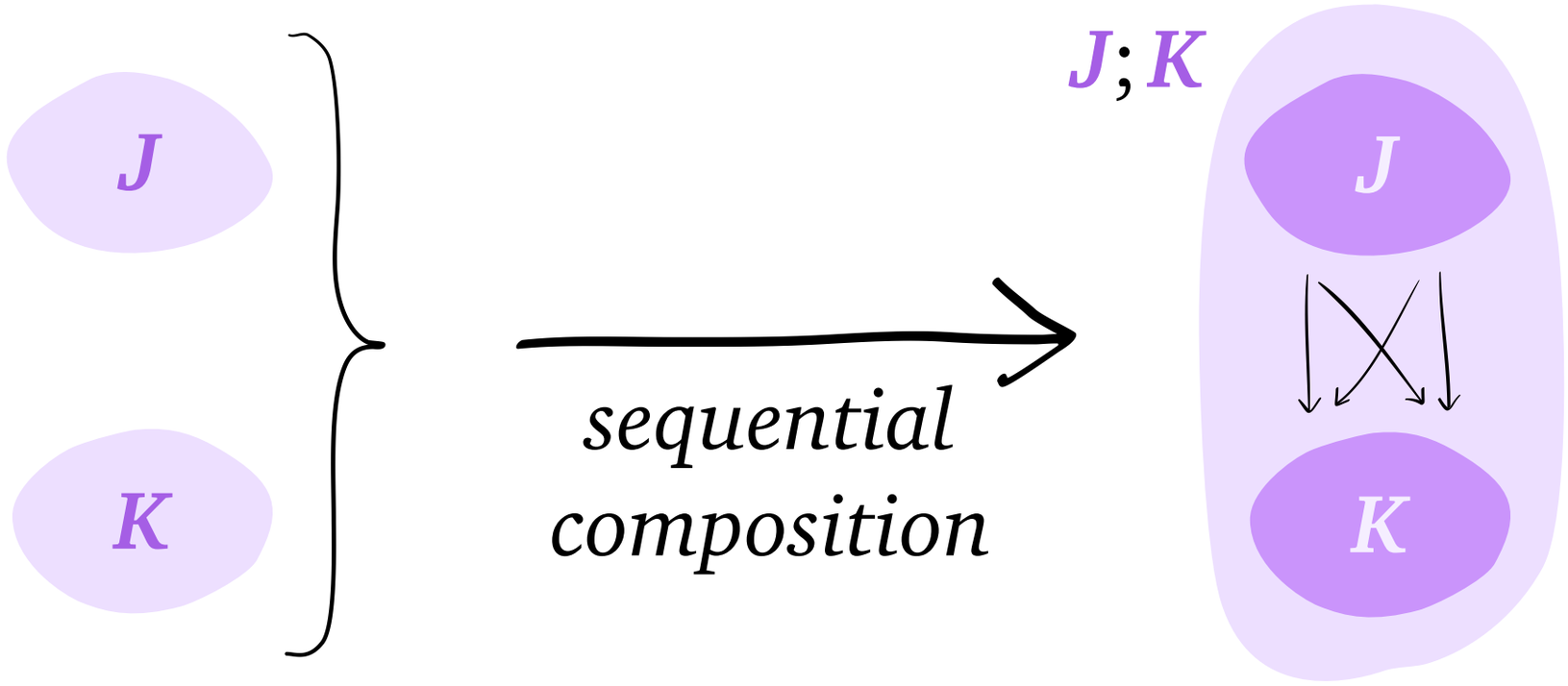
Splitting Big Jobs



Composition Law



Composition Law



Theorem (Composition Law):

$$V[\boldsymbol{r}](\boldsymbol{J};\boldsymbol{K}) = V[V[\boldsymbol{r}](\boldsymbol{K})](\boldsymbol{J})$$

Proof Idea

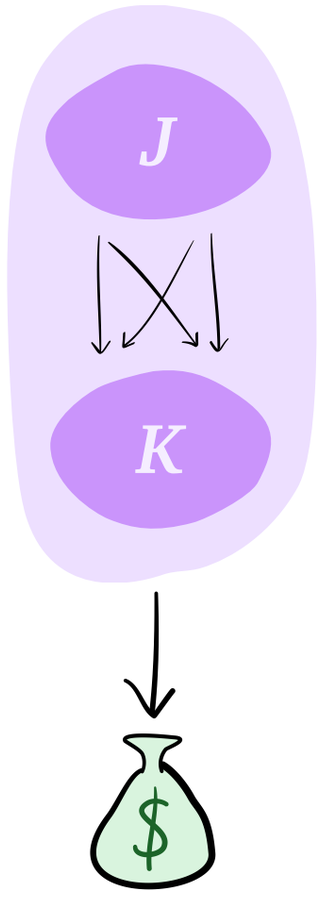
Theorem (Composition Law):

$$V[\boldsymbol{r}](\boldsymbol{J};\boldsymbol{K}) = V[V[\boldsymbol{r}](\boldsymbol{K})](\boldsymbol{J})$$

Proof Idea

Theorem (Composition Law):

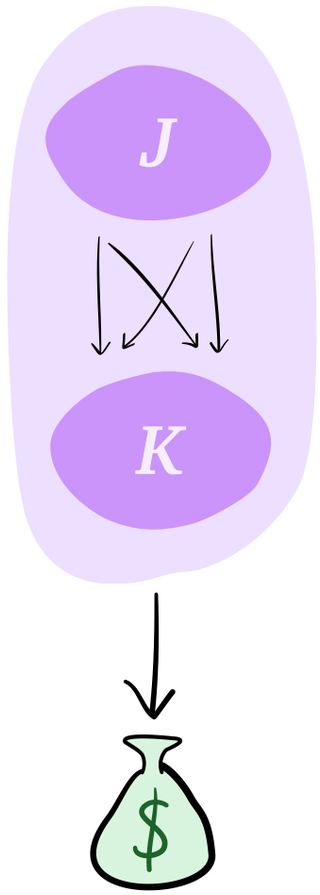
$$V[\boldsymbol{r}](\boldsymbol{J};\boldsymbol{K}) = V[V[\boldsymbol{r}](\boldsymbol{K})](\boldsymbol{J})$$

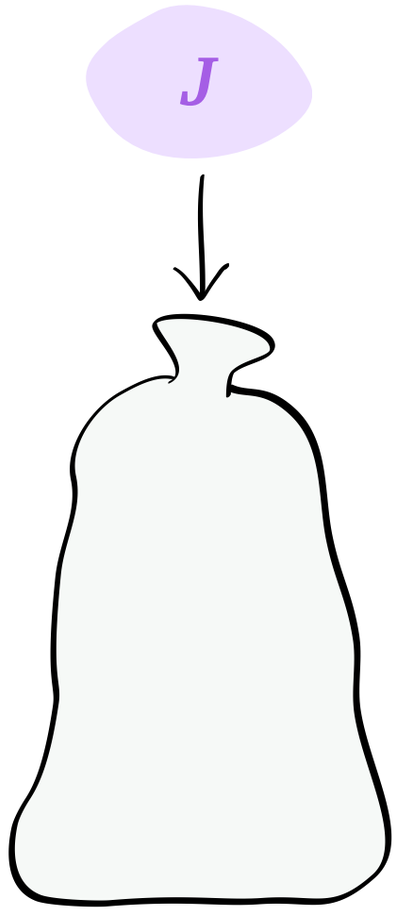


Proof Idea

Theorem (Composition Law):

$$V[\boldsymbol{r}](\boldsymbol{J};\boldsymbol{K}) = V[V[\boldsymbol{r}](\boldsymbol{K})](\boldsymbol{J})$$

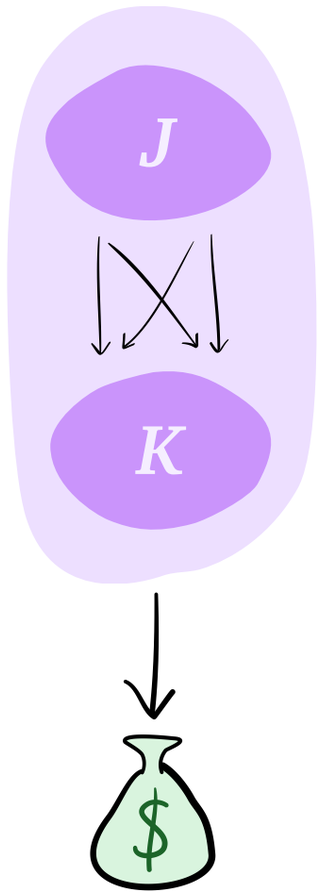


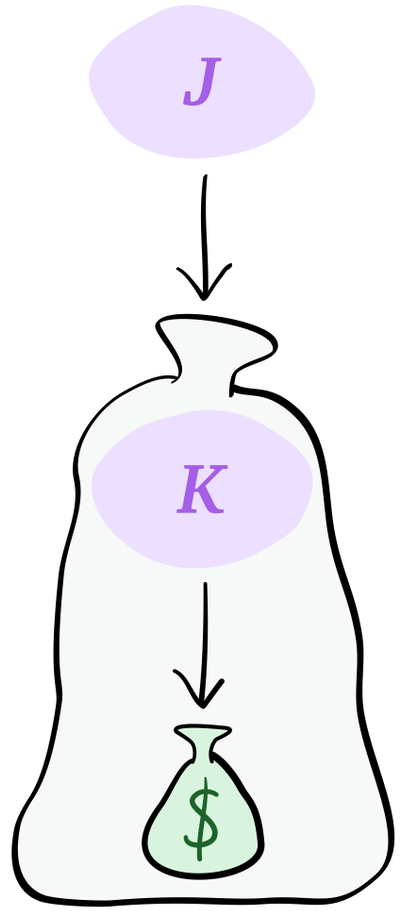


Proof Idea

Theorem (Composition Law):

$$V[\boldsymbol{r}](\boldsymbol{J};\boldsymbol{K}) = V[V[\boldsymbol{r}](\boldsymbol{K})](\boldsymbol{J})$$

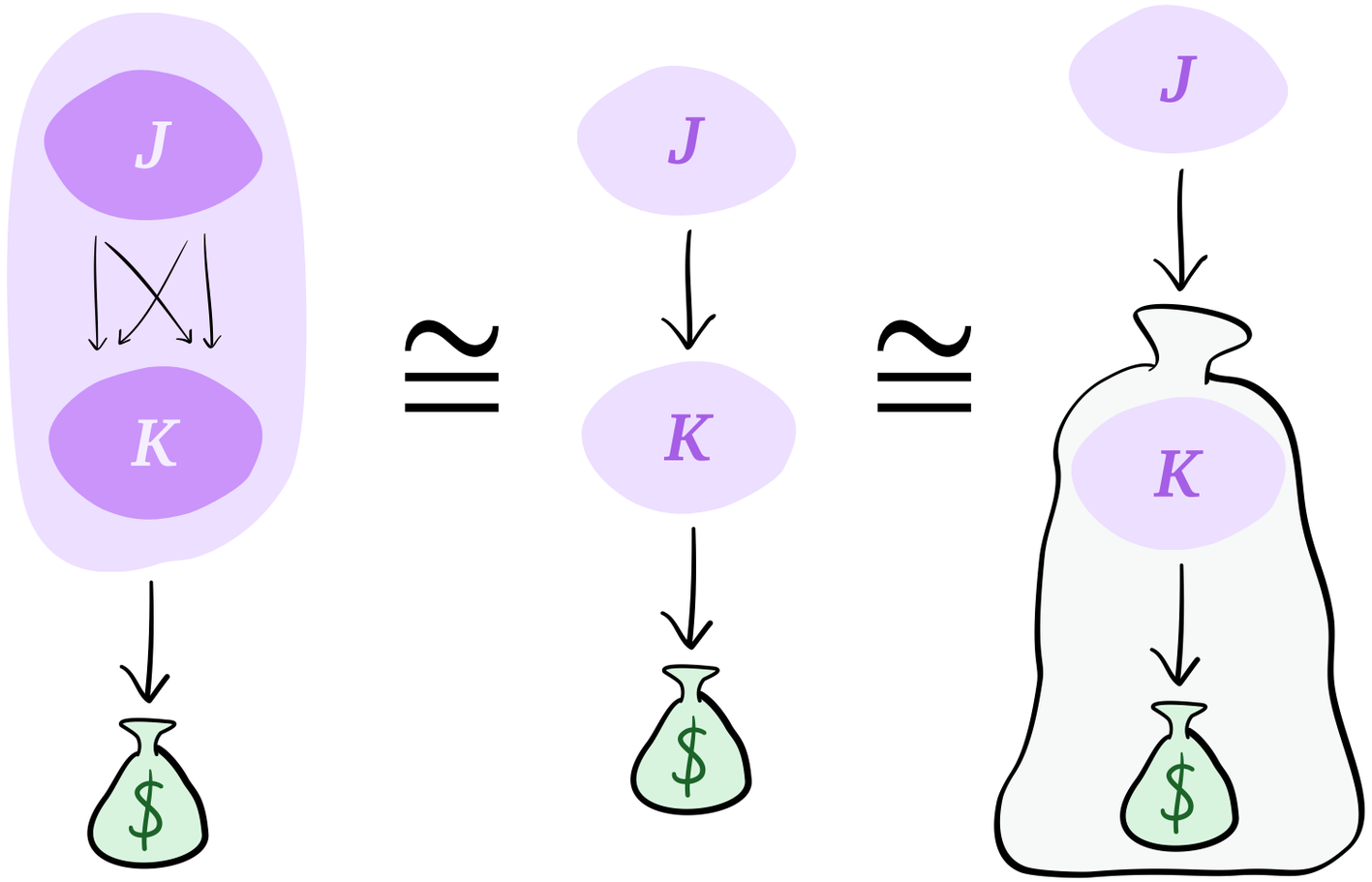




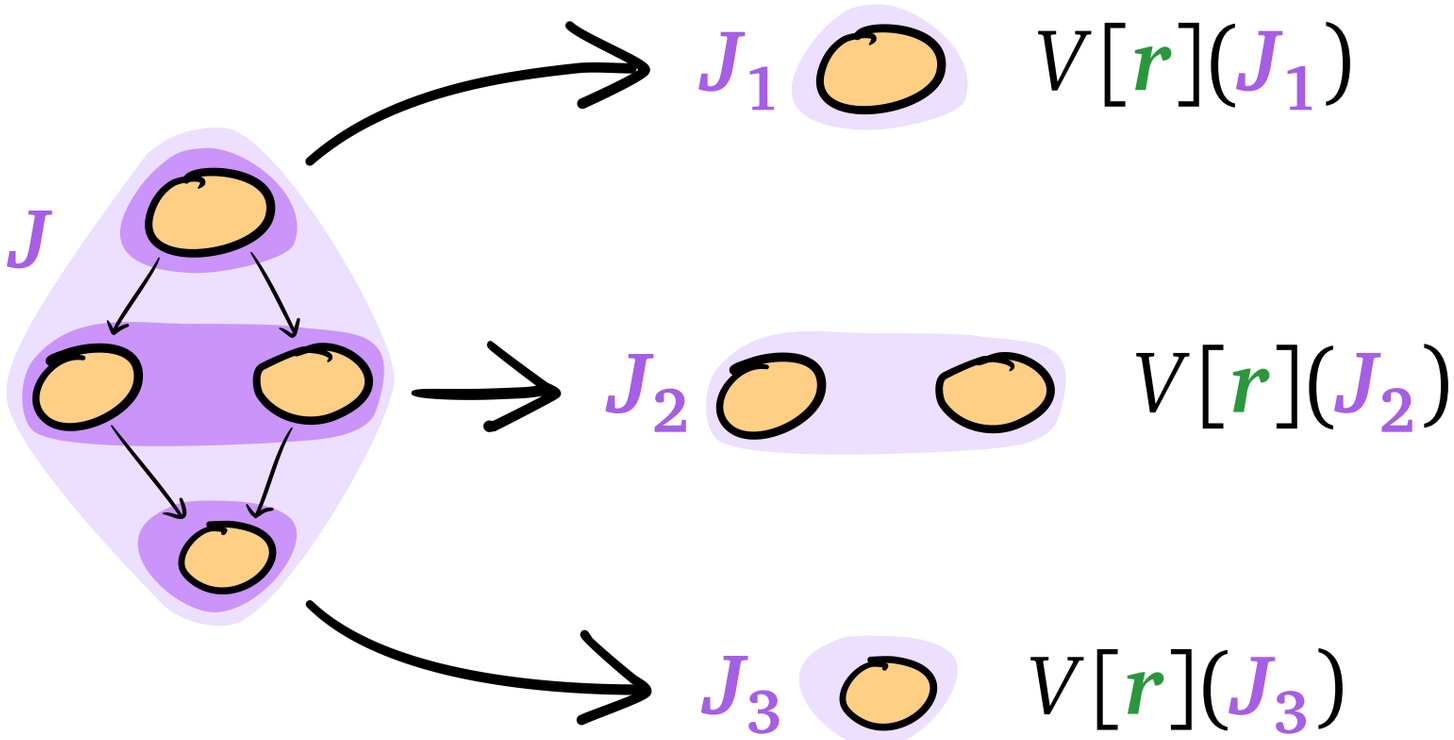
Proof Idea

Theorem (Composition Law):

$$V[\boldsymbol{r}](\boldsymbol{J};\boldsymbol{K}) = V[V[\boldsymbol{r}](\boldsymbol{K})](\boldsymbol{J})$$

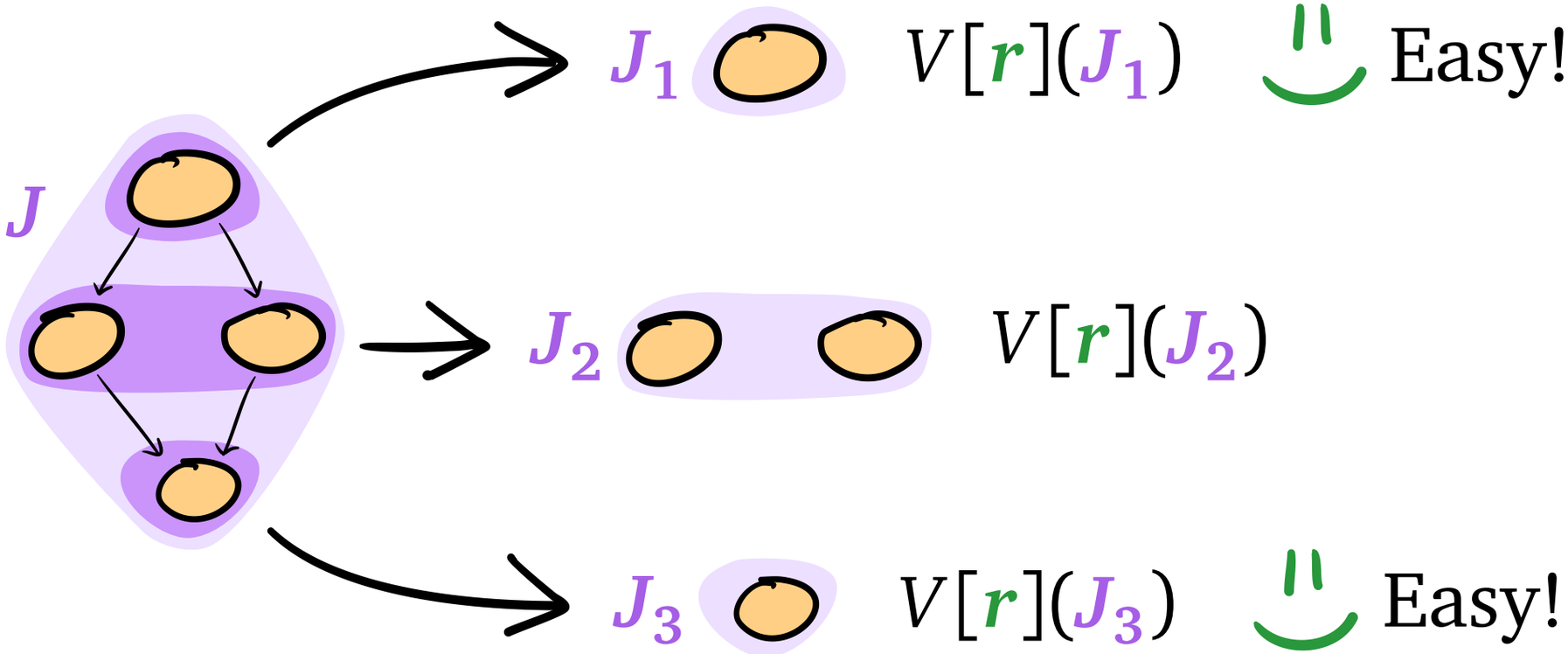


Applying Composition

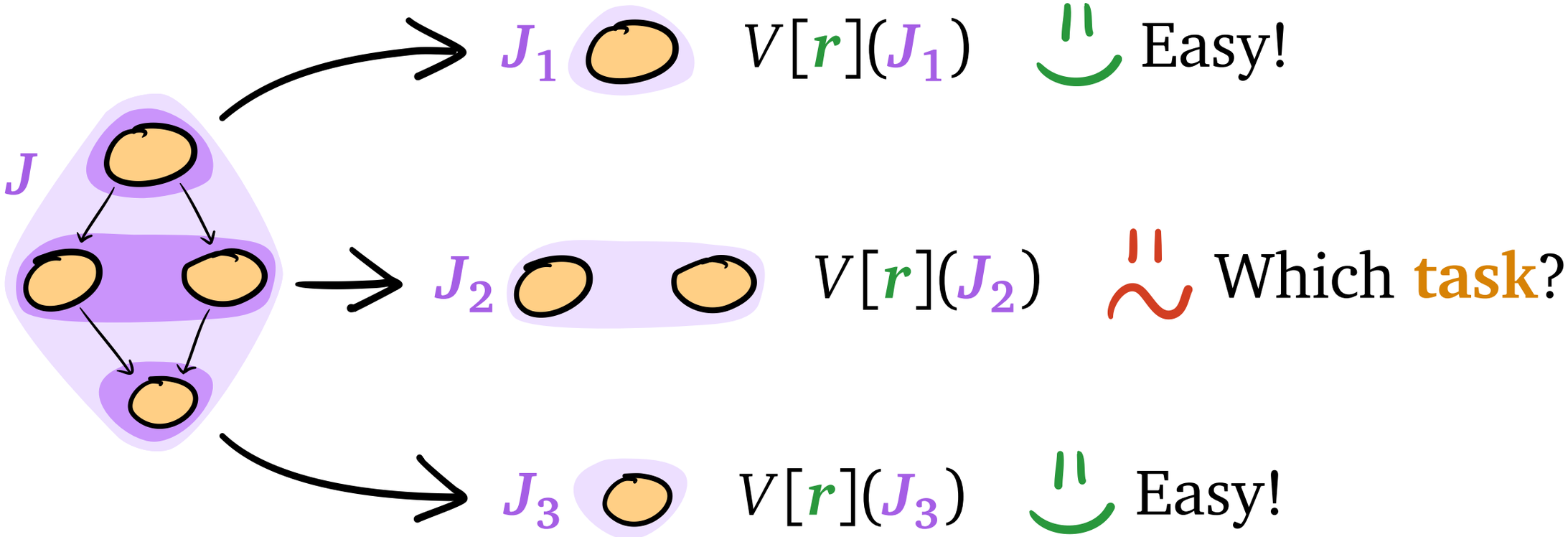


phases

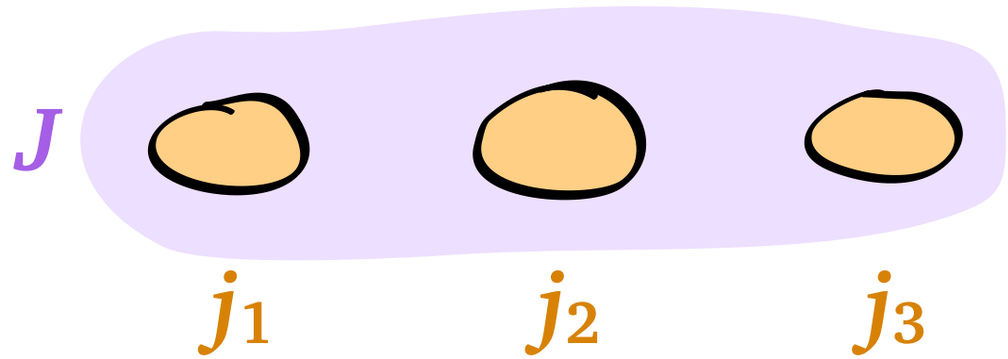
Applying Composition



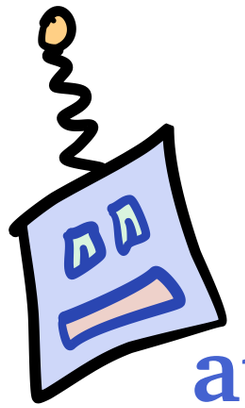
Applying Composition



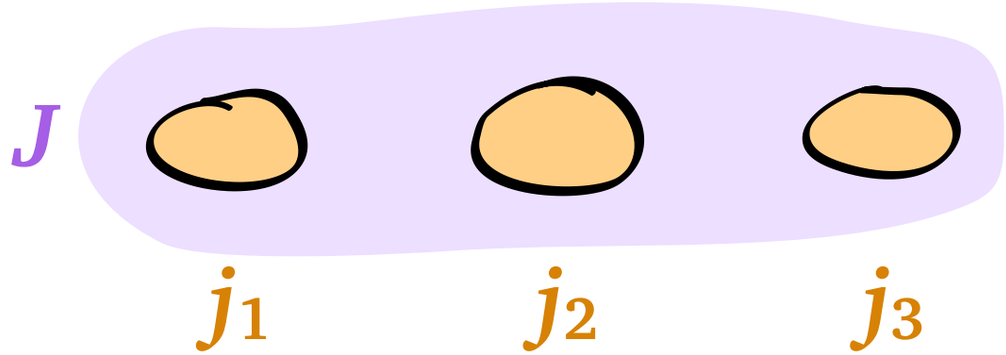
Autopiloting Law



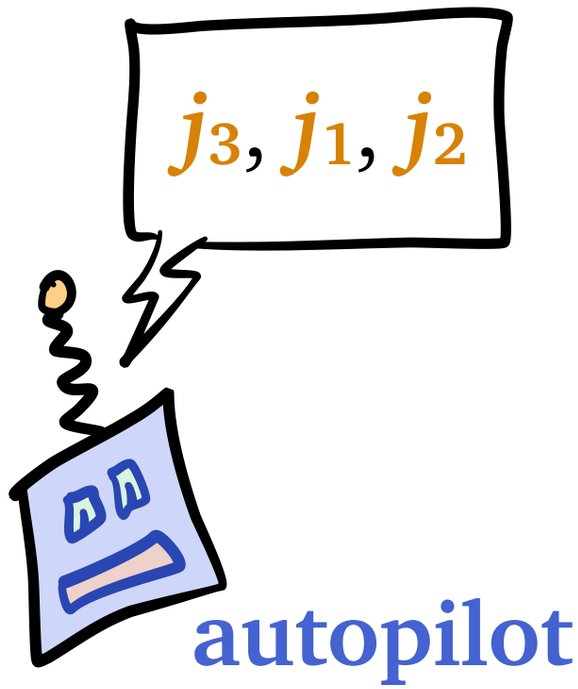
Autopiloting Law

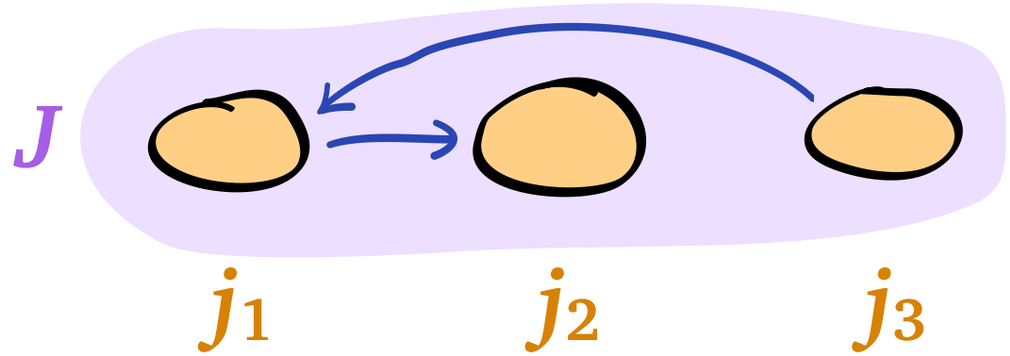


autopilot

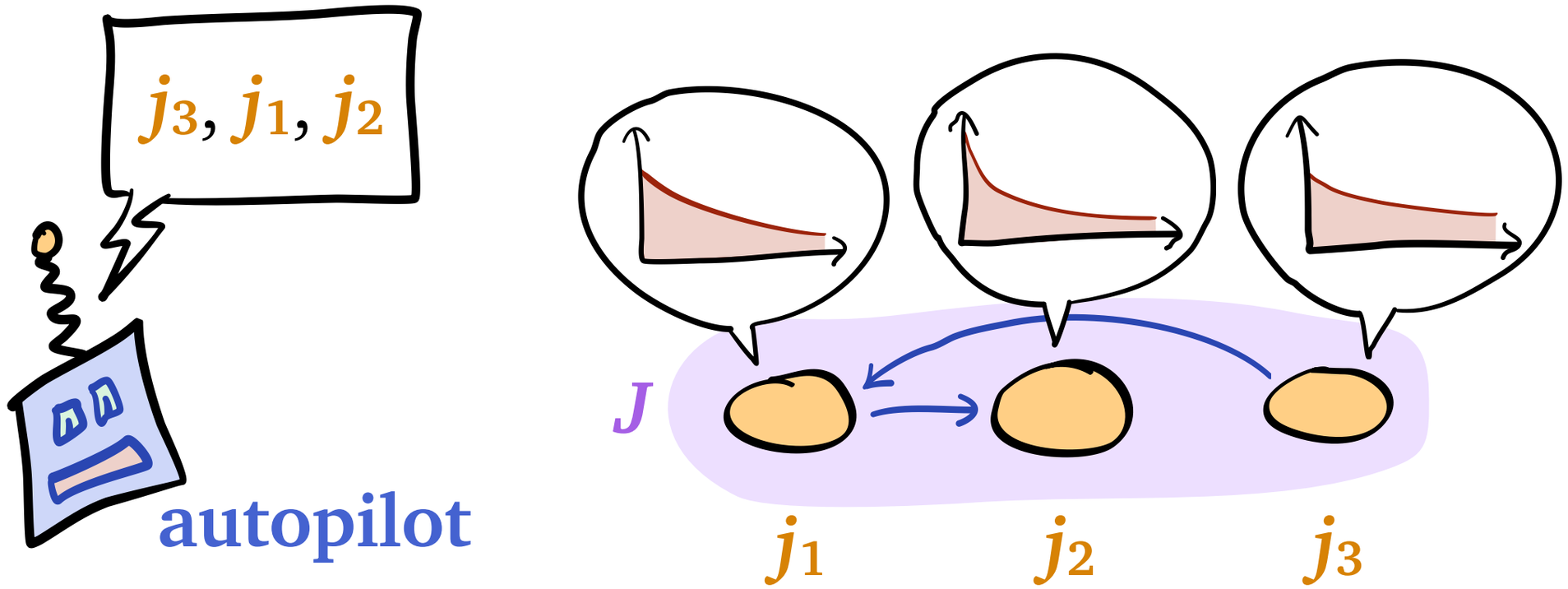


Autopiloting Law



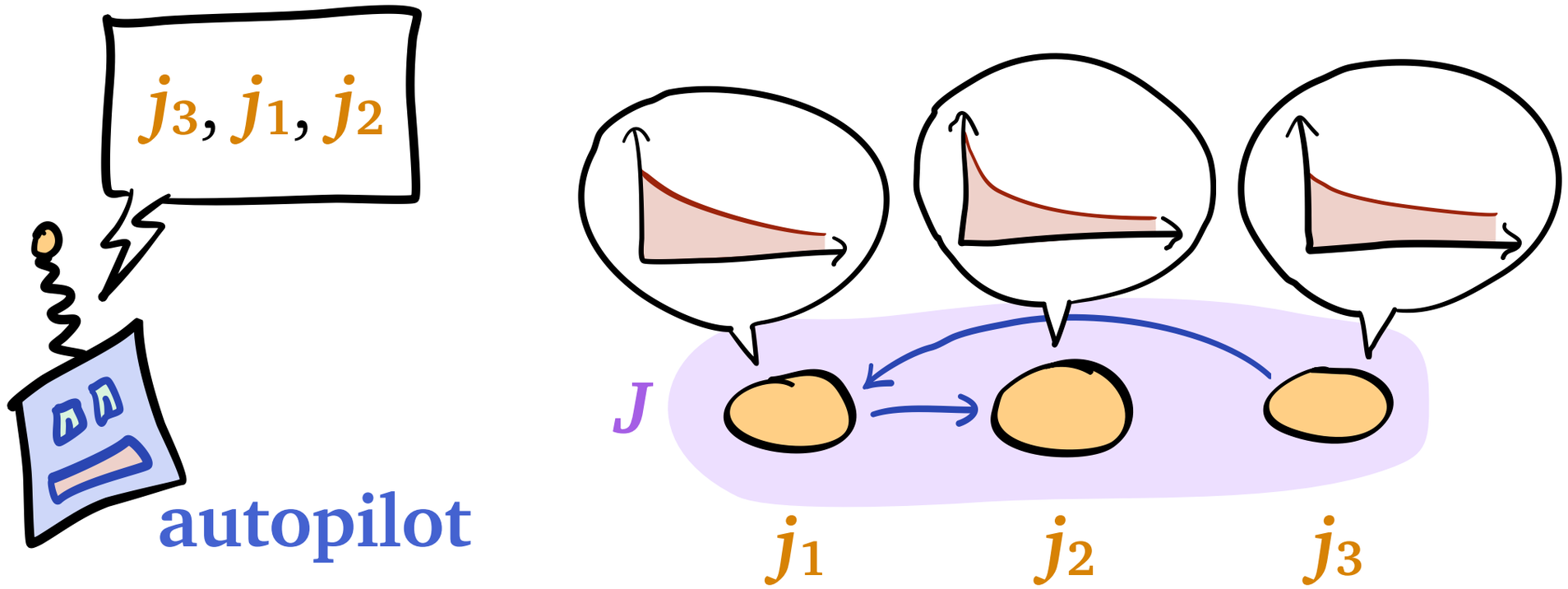


Autopiloting Law



Theorem (Autopiloting Law): any fully parallel job with **Pareto** tasks of same α has an **autopilot**

Autopiloting Law



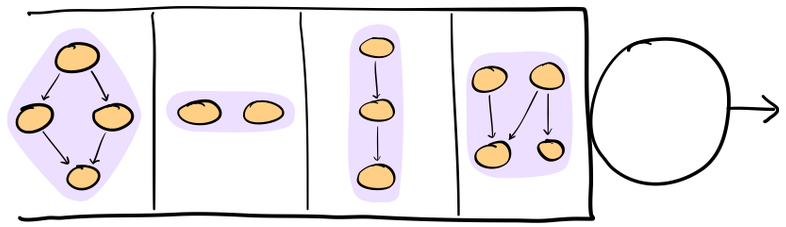
Theorem (Autopiloting Law): any fully parallel job with **Pareto** tasks of same α has an **autopilot**

- serves task that is *longest in expectation*

Summary

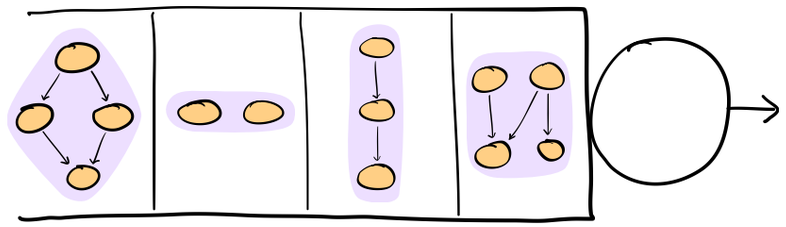
Summary

Goal: multitask scheduling

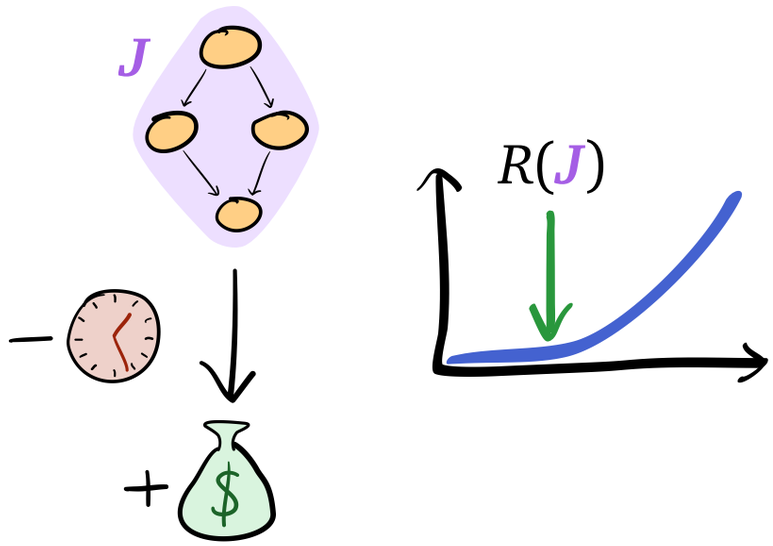


Summary

Goal: multitask scheduling

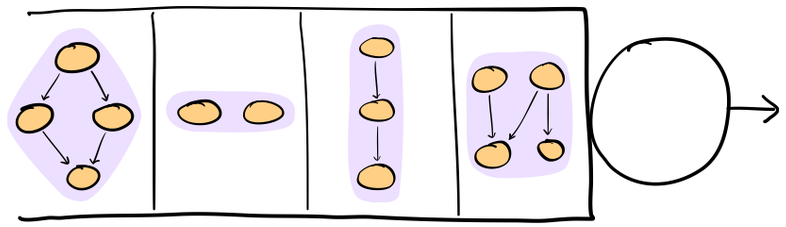


Approach: single-job profit



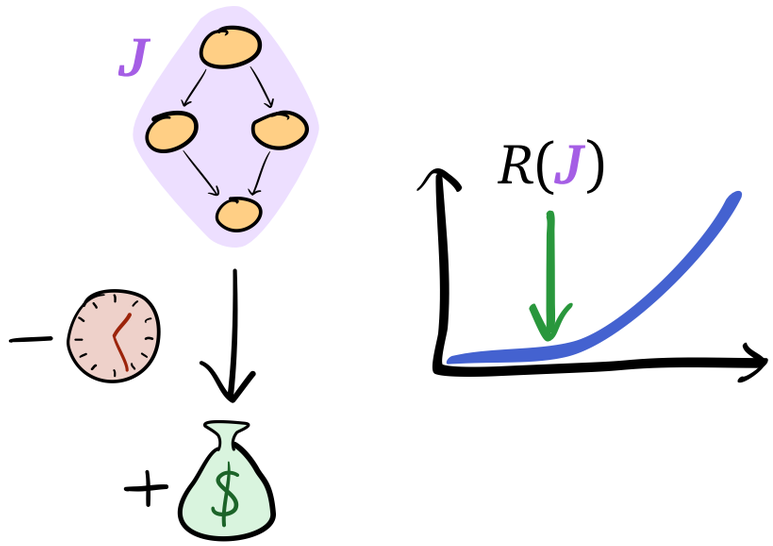
Summary

Goal: multitask scheduling



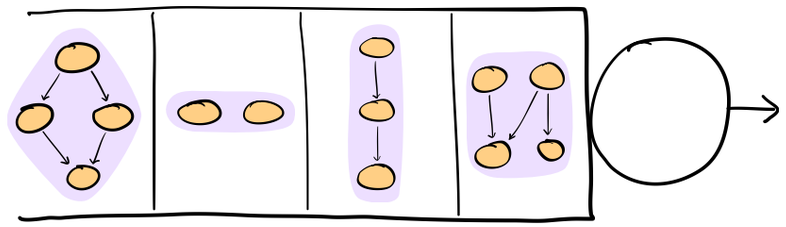
Obstacle: computation

Approach: single-job profit

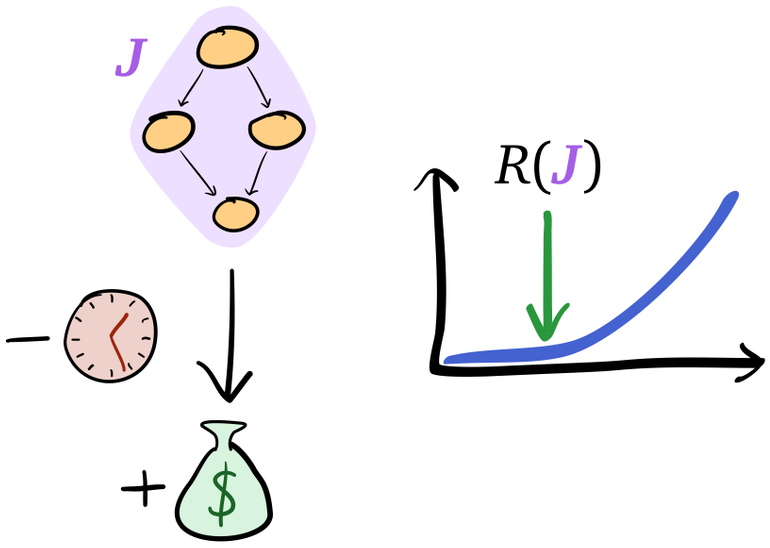


Summary

Goal: multitask scheduling

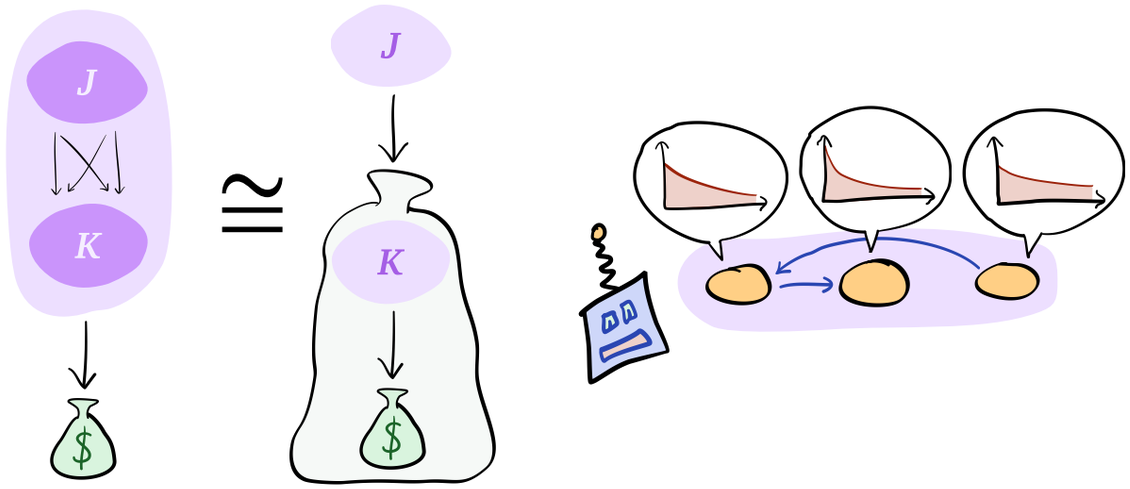


Approach: single-job profit



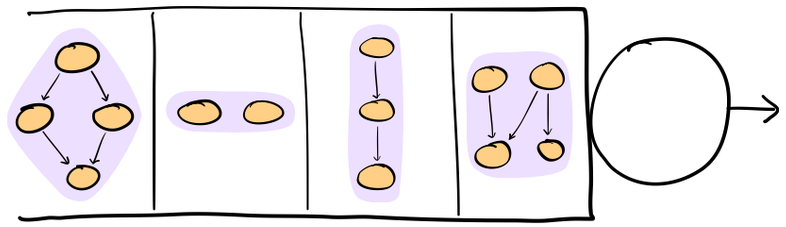
Obstacle: computation

Solutions: composition law
autopiloting law

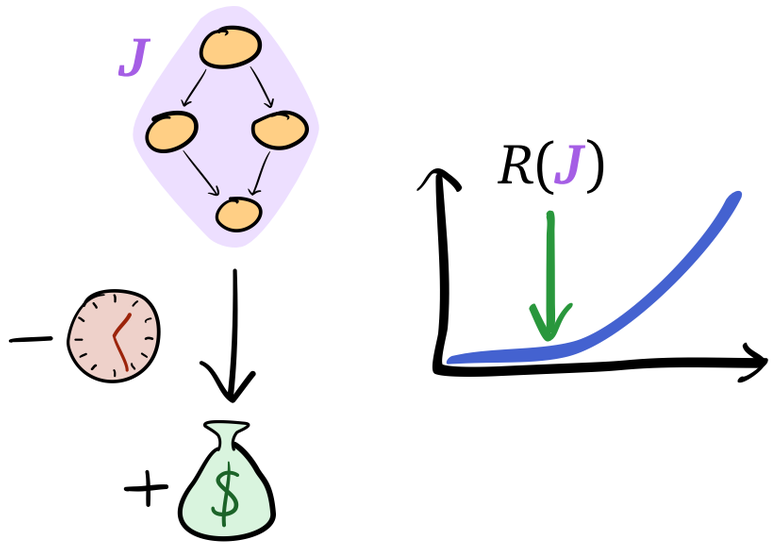


Summary

Goal: multitask scheduling

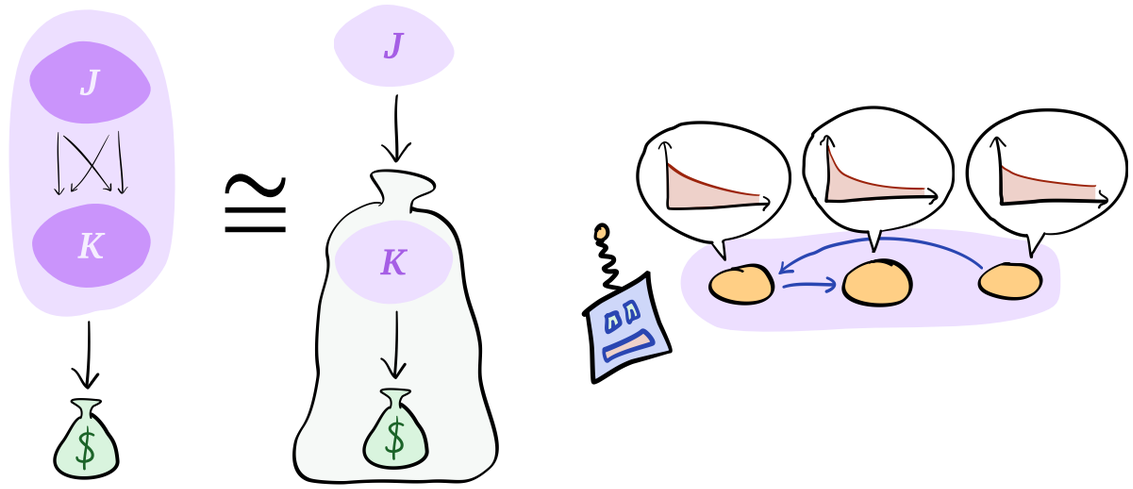


Approach: single-job profit



Obstacle: computation

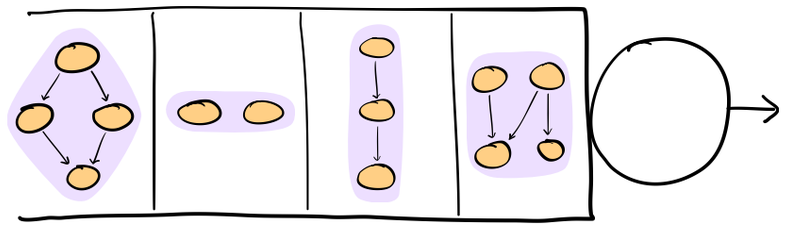
Solutions: composition law
autopiloting law



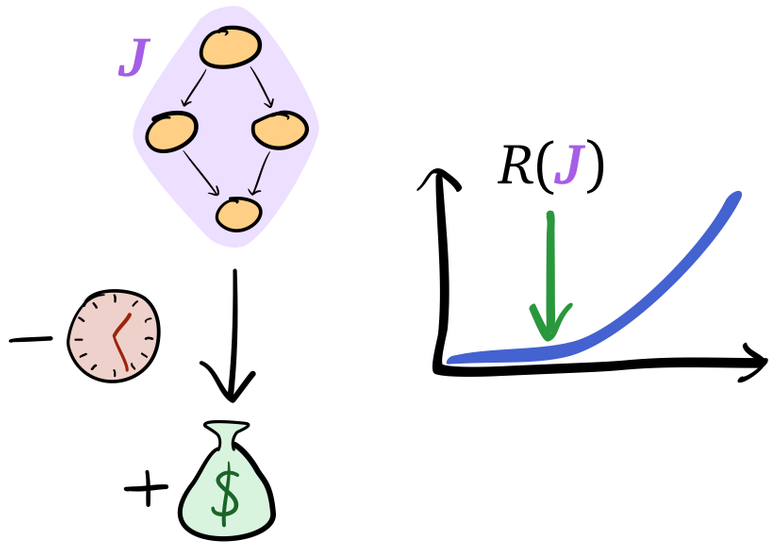
Obstacle: proving optimality

Summary

Goal: multitask scheduling

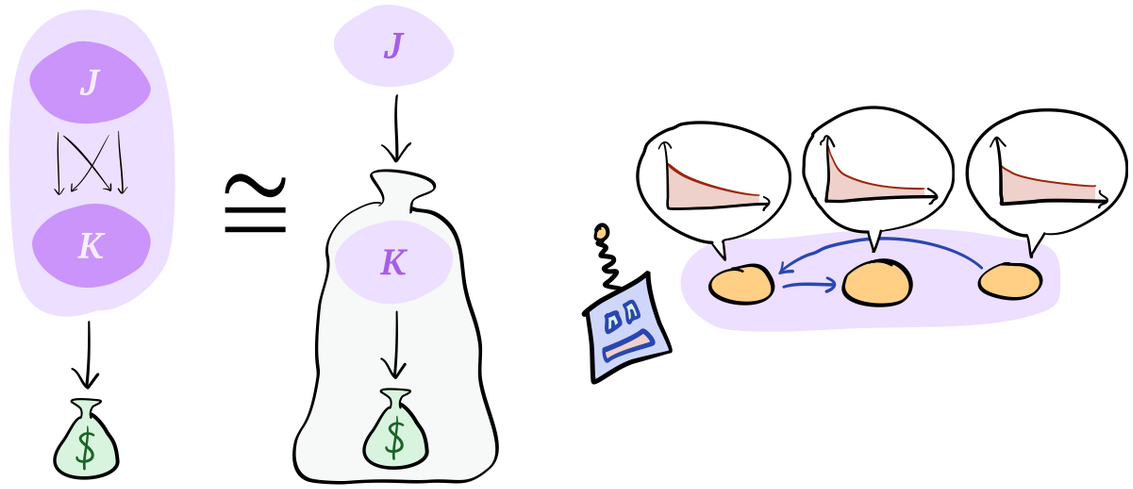


Approach: single-job profit



Obstacle: computation

Solutions: composition law
autopiloting law



Obstacle: proving optimality

Solution: autopiloting law