Karma:
Resource Allocation for Dynamic Demands


Midhul Vuppalapati


Giannis Fikioris


Rachit Agarwal


Asaf Cidon


Anurag Khandelwal


Éva Tardos

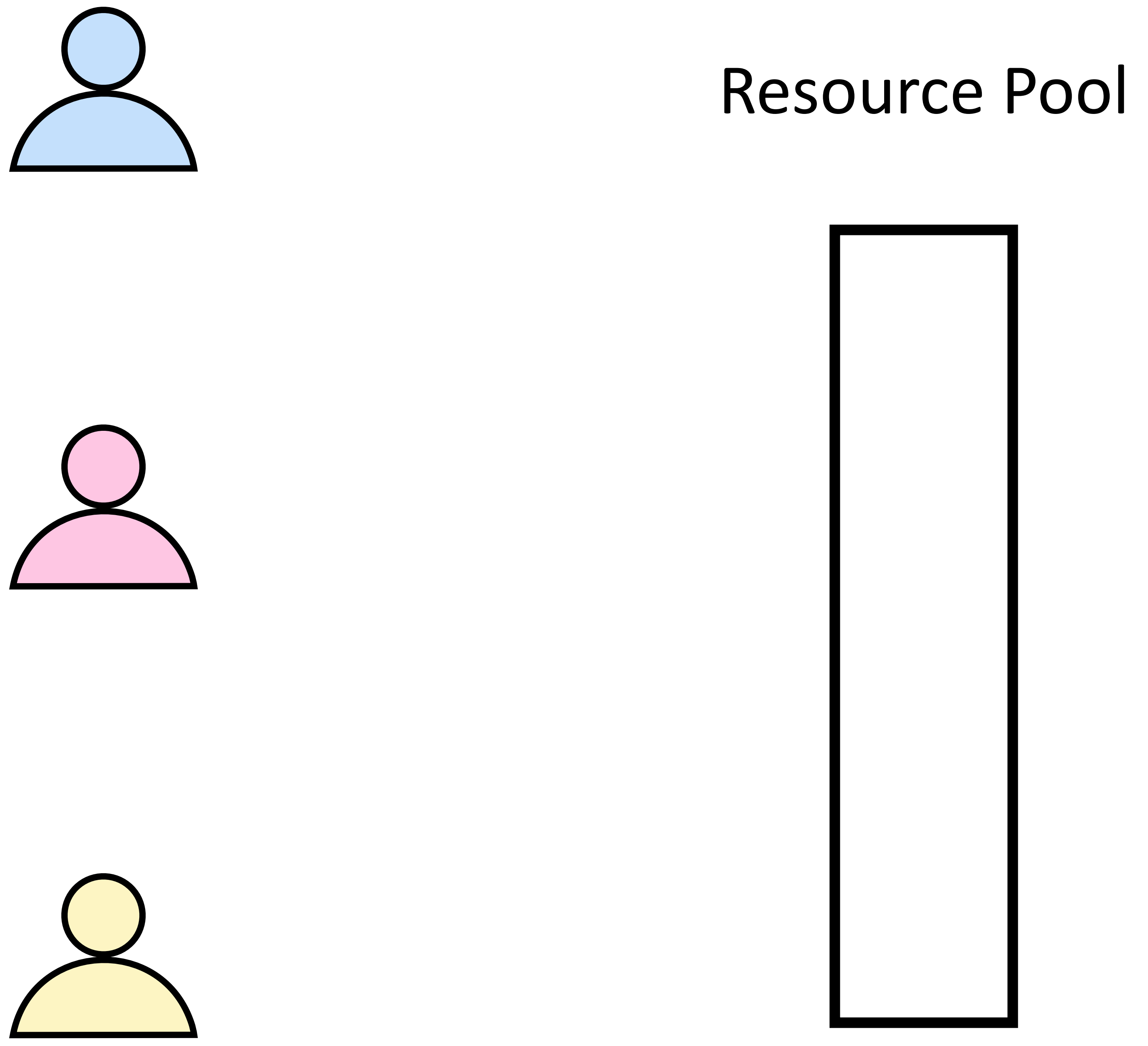
Cornell University

Columbia University

Yale University

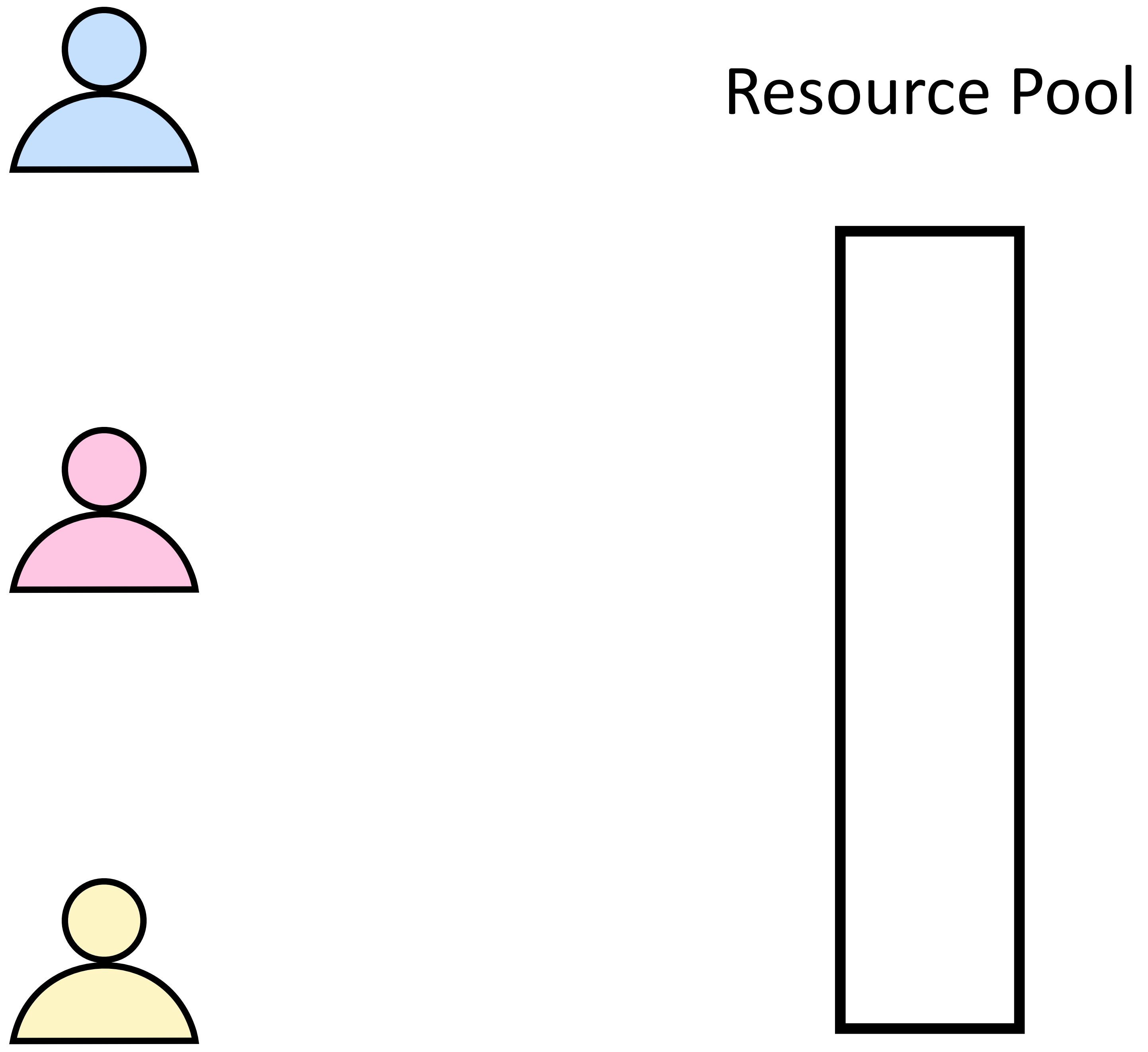
Resource allocation is a fundamental problem

Allocating a resource (e.g. CPU, Memory) with a fixed capacity across multiple users



Resource allocation is a fundamental problem

Allocating a resource (e.g. CPU, Memory) with a fixed capacity across multiple users

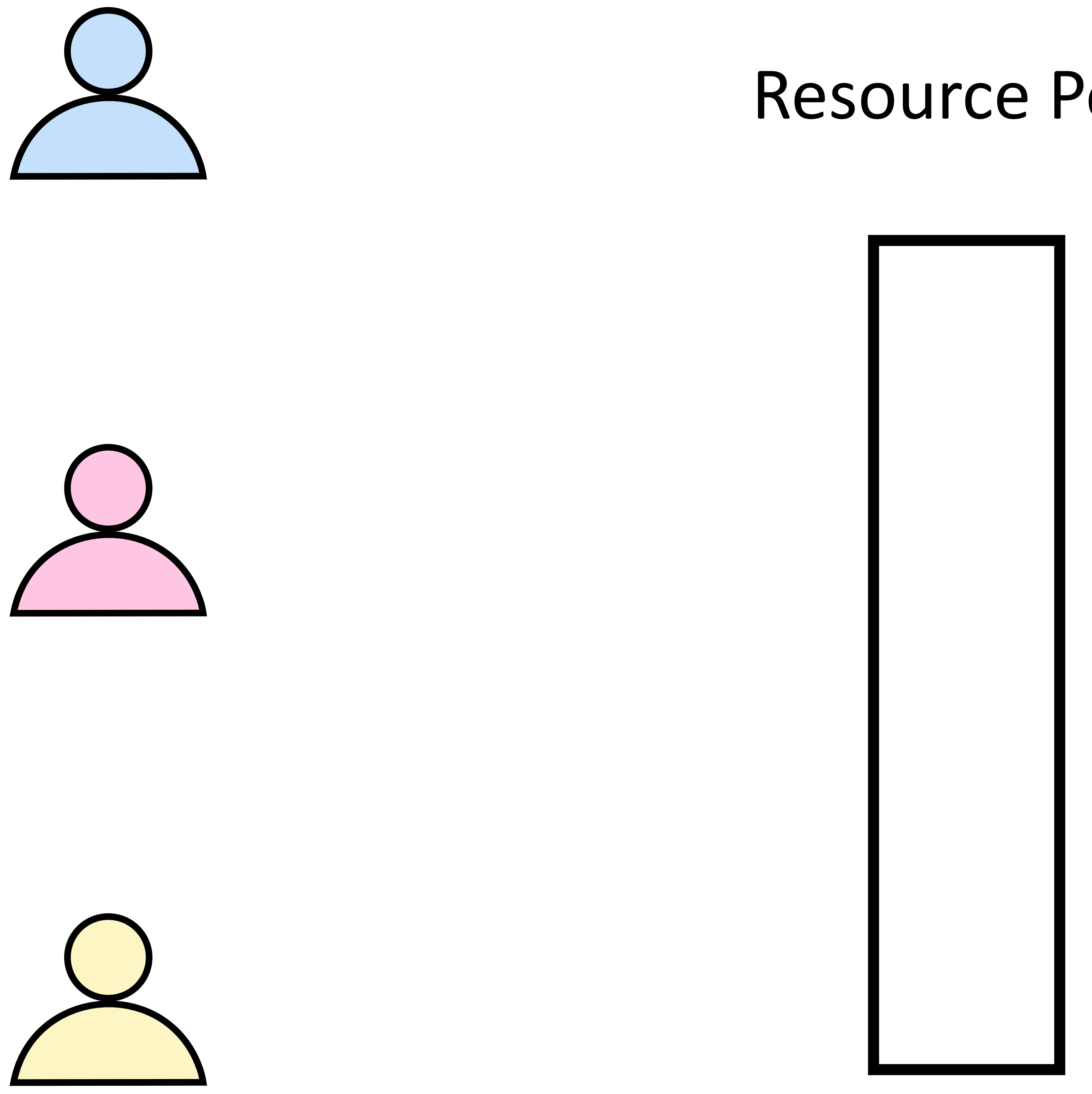


Key Desirable Properties

Resource allocation is a fundamental problem

Allocating a resource (e.g. CPU, Memory) with a fixed capacity across multiple users

Resource Pool



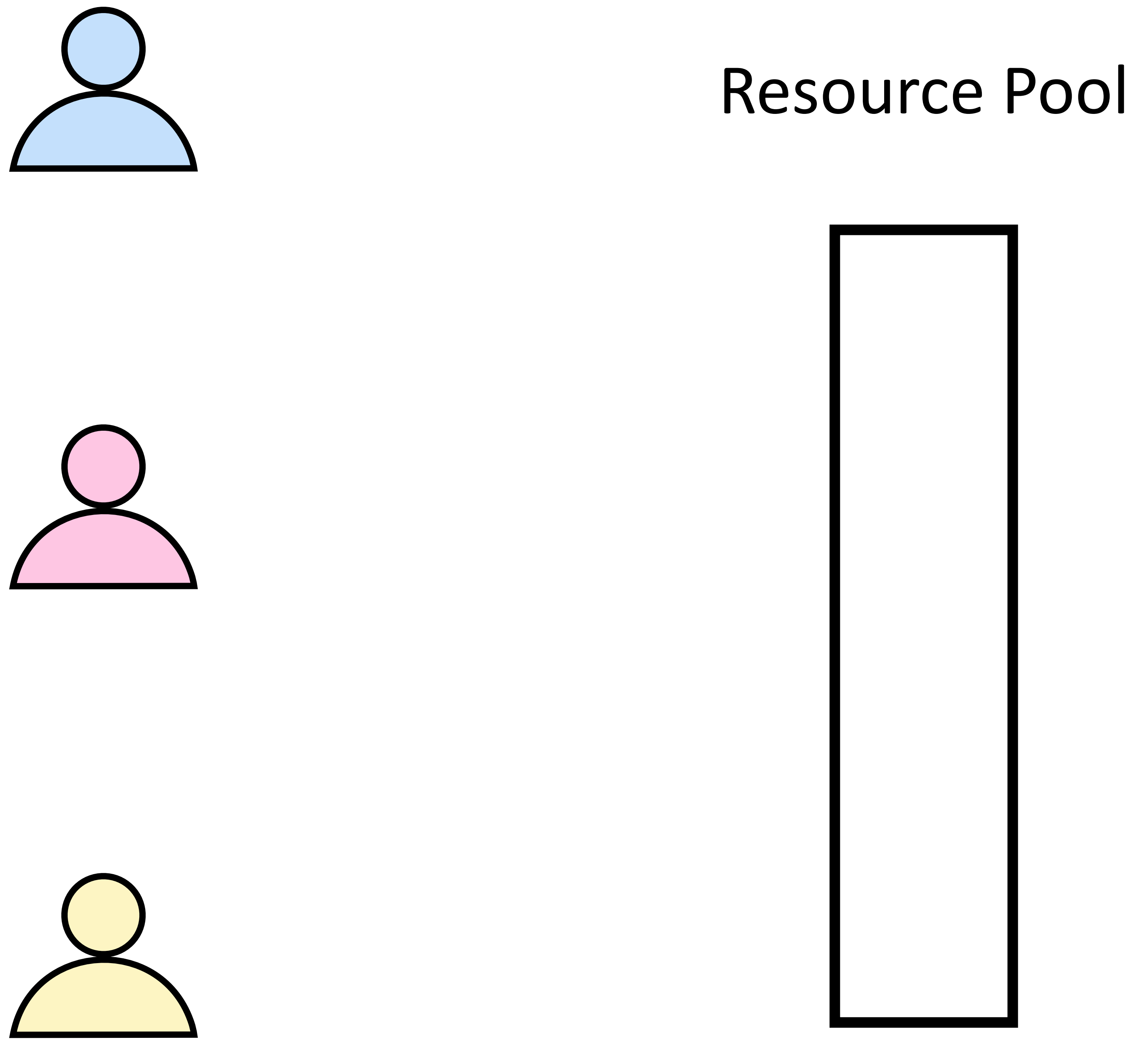
Key Desirable Properties

Pareto efficiency

Resources should not remain unused if there is demand

Resource allocation is a fundamental problem

Allocating a resource (e.g. CPU, Memory) with a fixed capacity across multiple users



Key Desirable Properties

Pareto efficiency

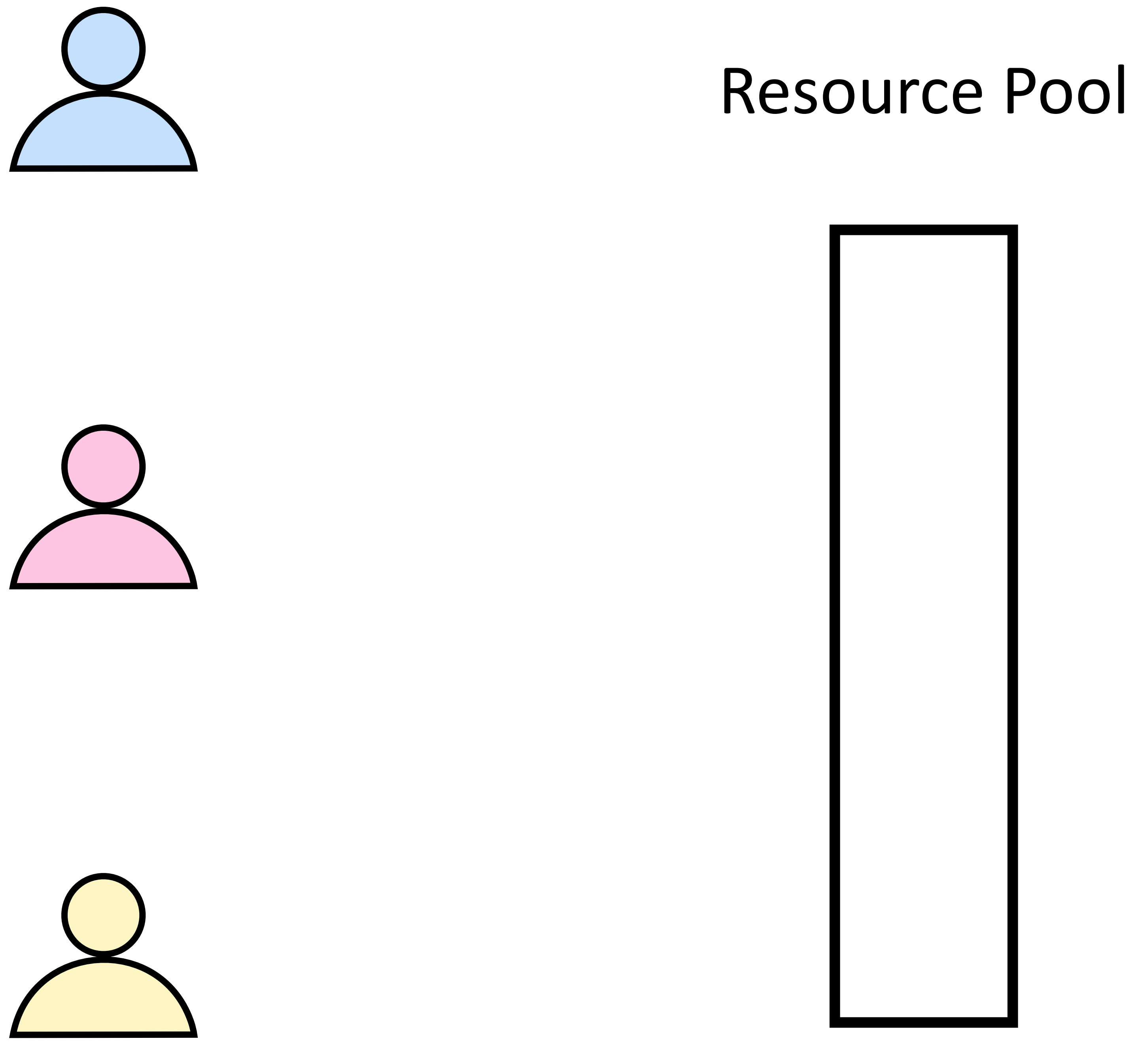
Resources should not remain unused if there is demand

Strategy-proofness

Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)

Resource allocation is a fundamental problem

Allocating a resource (e.g. CPU, Memory) with a fixed capacity across multiple users



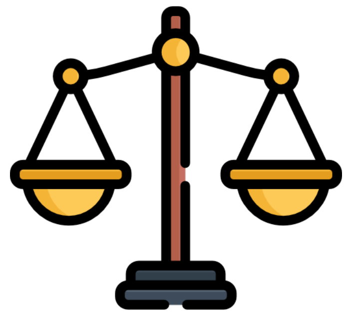
Key Desirable Properties

Pareto efficiency

Resources should not remain unused if there is demand

Strategy-proofness

Selfish users cannot increase their allocation by lying (selfish ≠ adversarial)

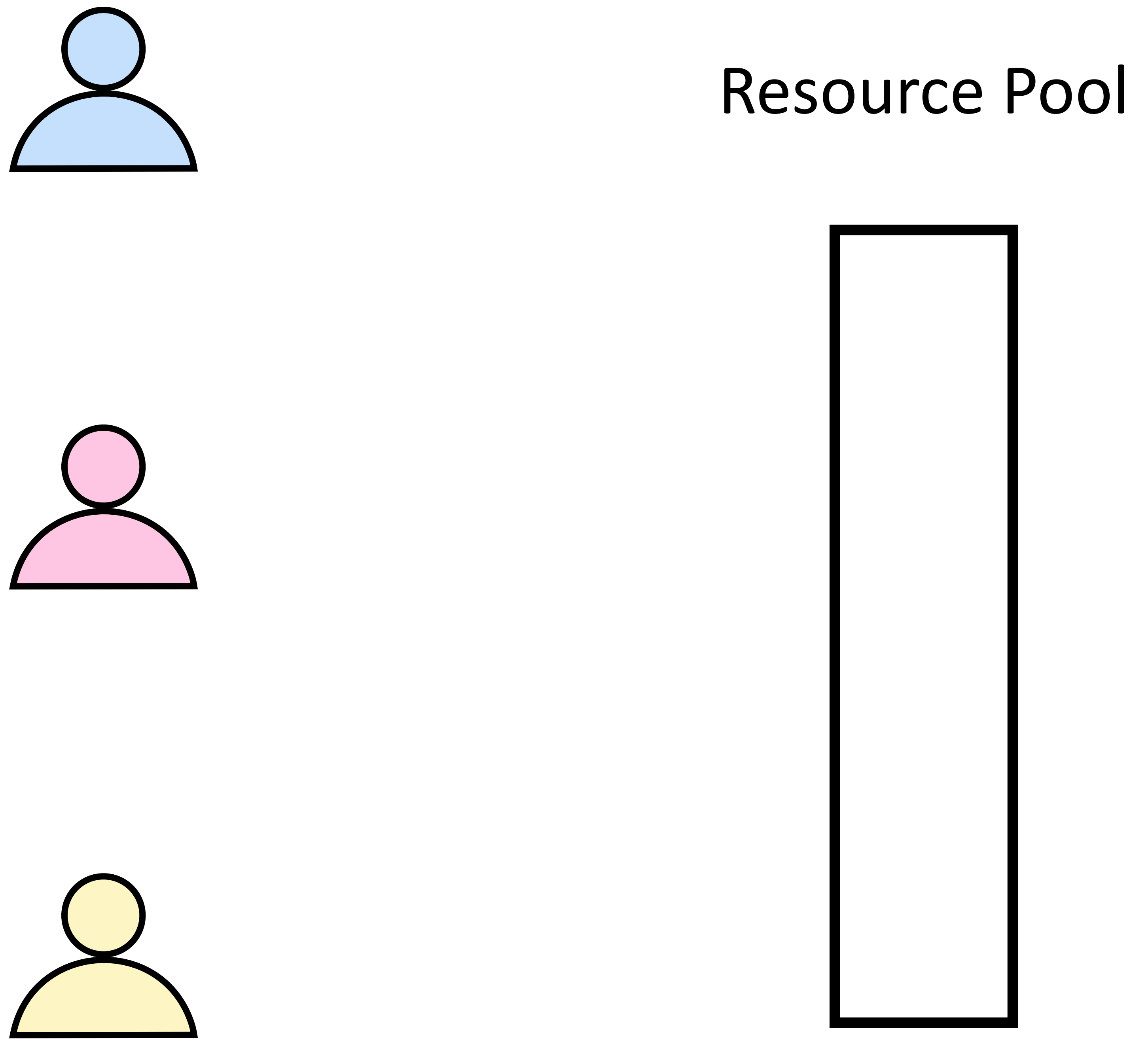


Fairness

Balanced resource allocations across users

Resource allocation is a fundamental problem

Allocating a resource (e.g. CPU, Memory) with a fixed capacity across multiple users



Key Desirable Properties

Pareto efficiency

Resources should not remain unused if there is demand

Strategy-proofness

Selfish users cannot increase their allocation by lying (selfish ≠ adversarial)

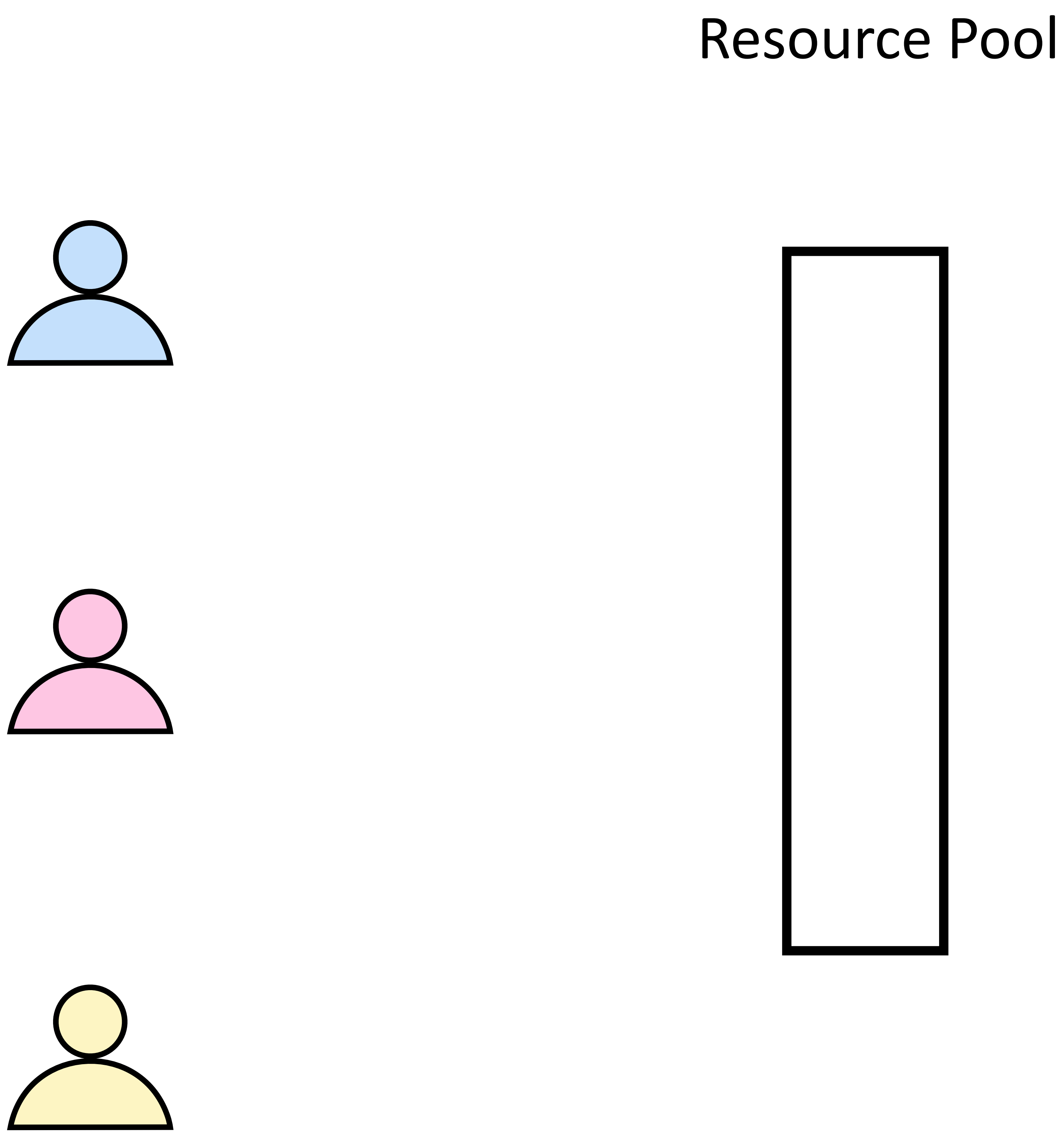
Fairness

Balanced resource allocations across users

Two popular resource allocation mechanisms (for single resource type): strict partitioning and max-min fairness

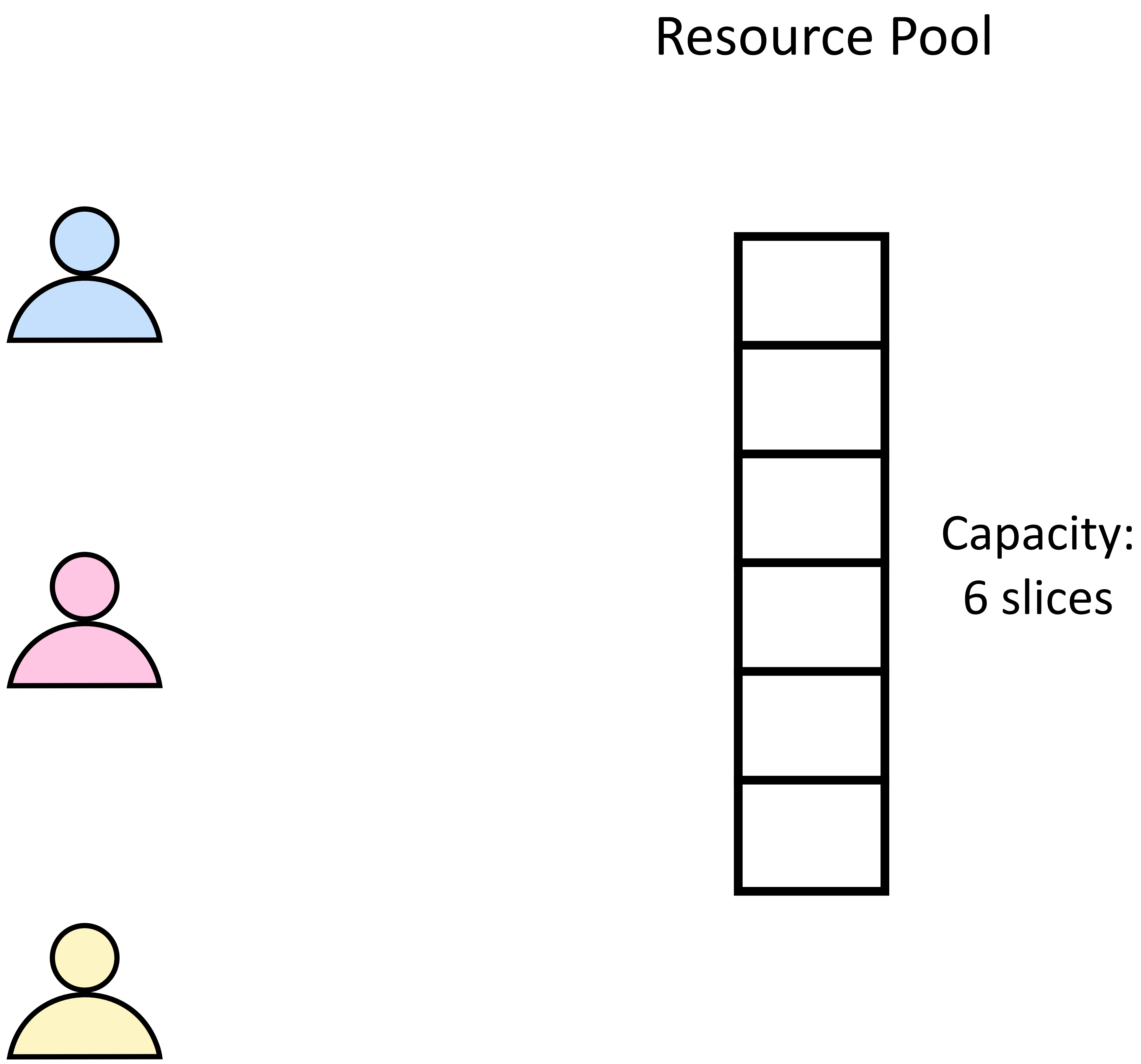
Strict partitioning

Allocating the resource equally across all users (“fair share”) **independent of their demands**



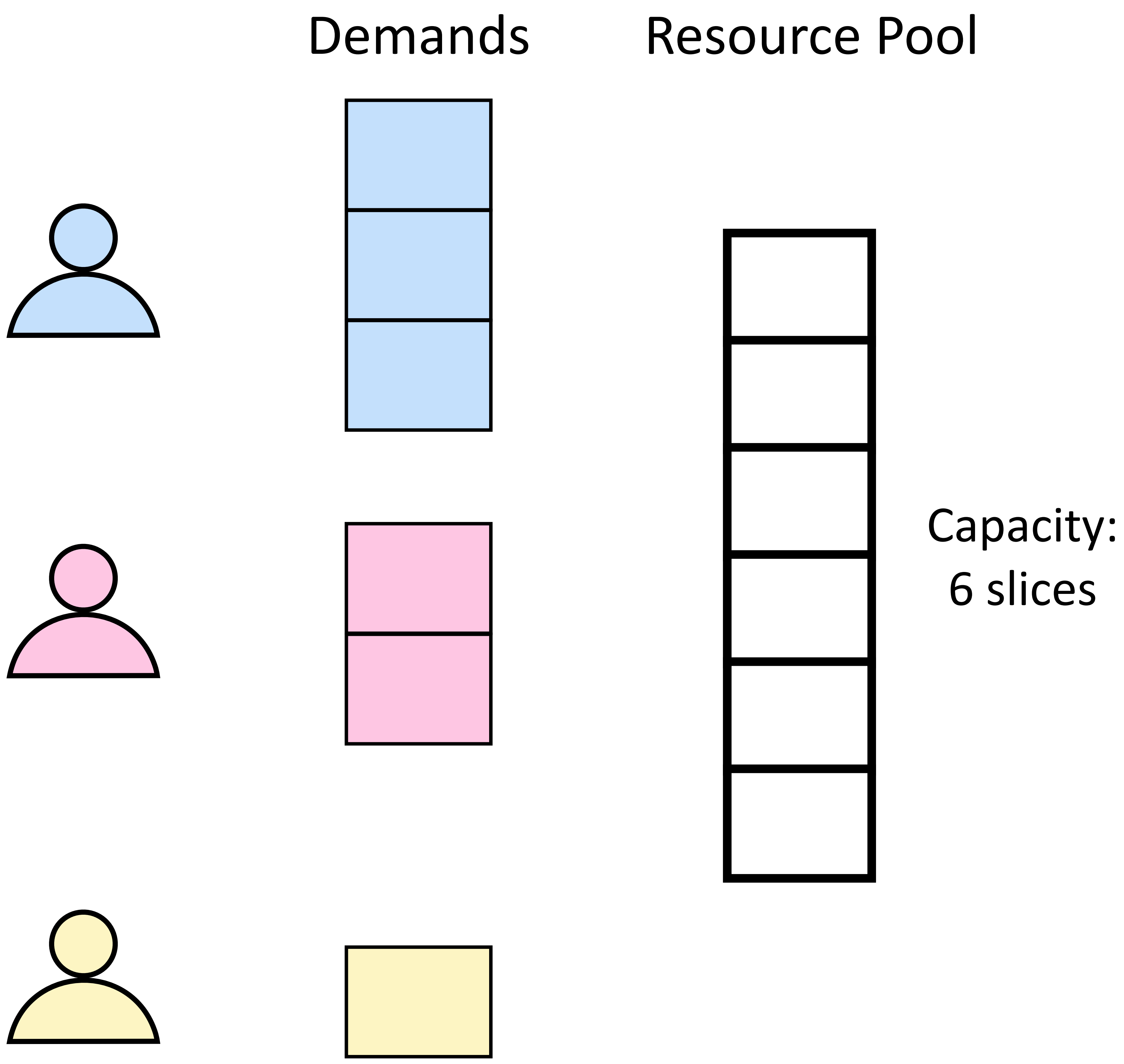
Strict partitioning

Allocating the resource equally across all users (“fair share”) **independent of their demands**



Strict partitioning

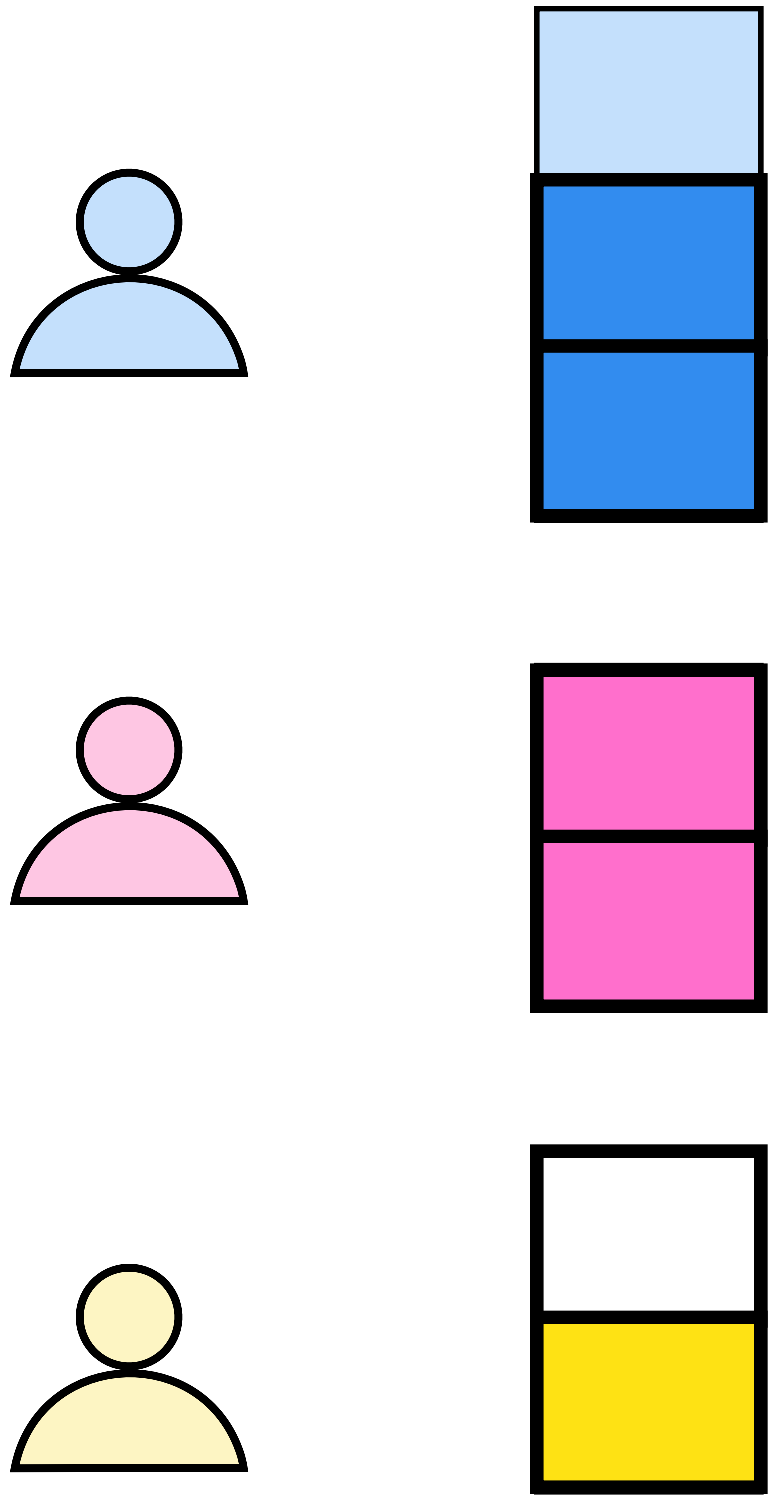
Allocating the resource equally across all users (“fair share”) **independent of their demands**



Strict partitioning

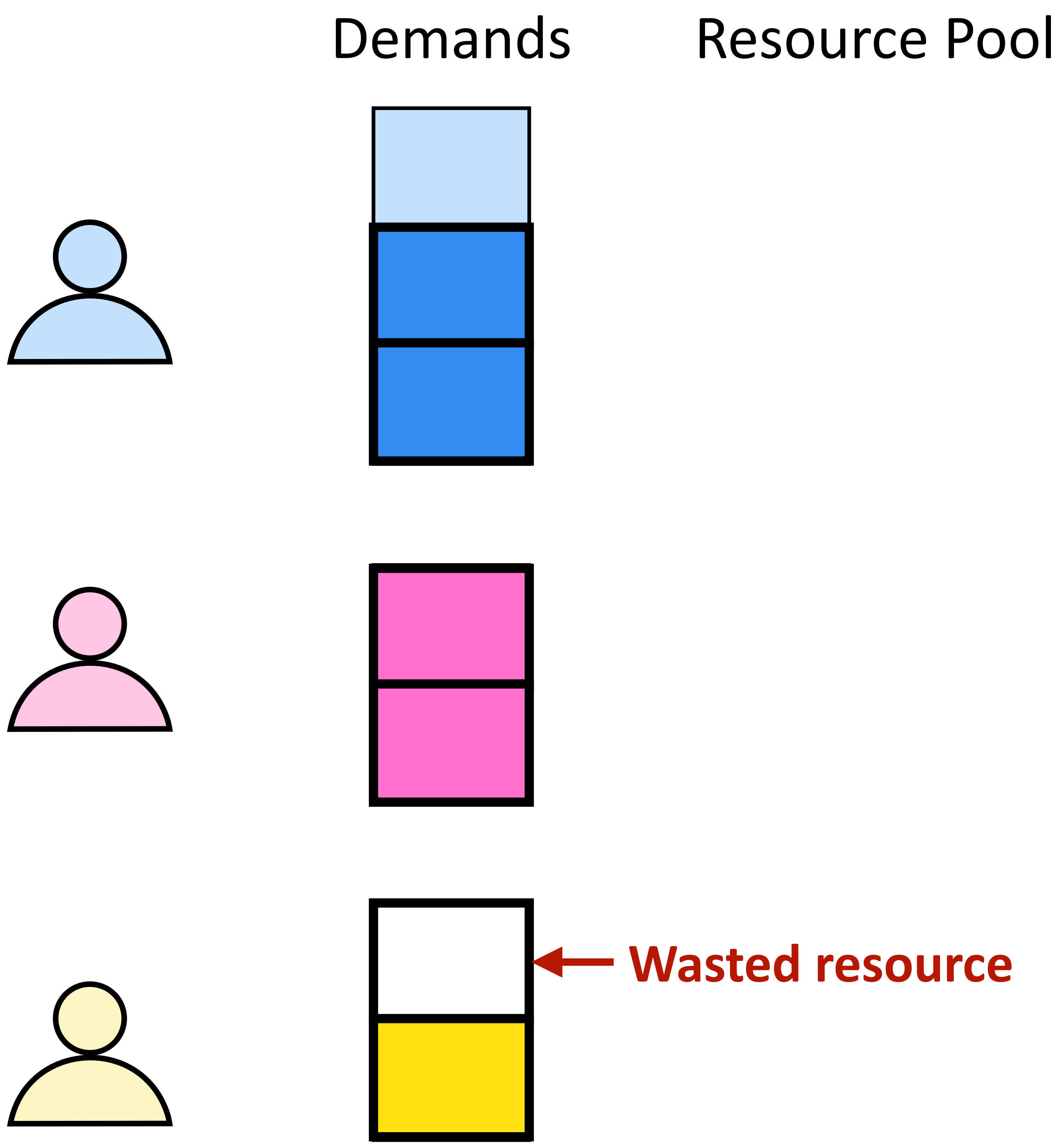
Allocating the resource equally across all users ("fair share") **independent of their demands**

Demands Resource Pool



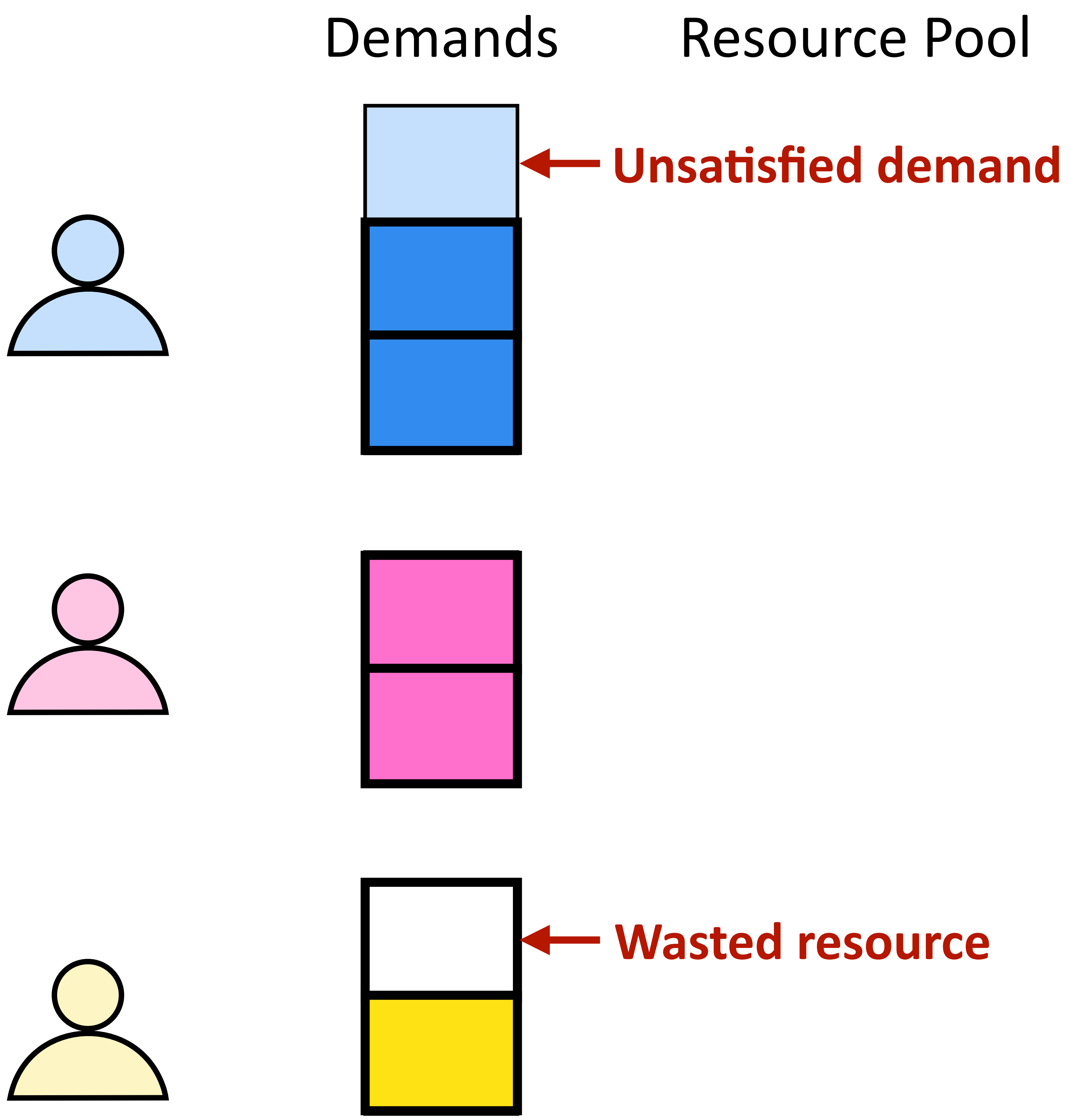
Strict partitioning

Allocating the resource equally across all users ("fair share") **independent of their demands**



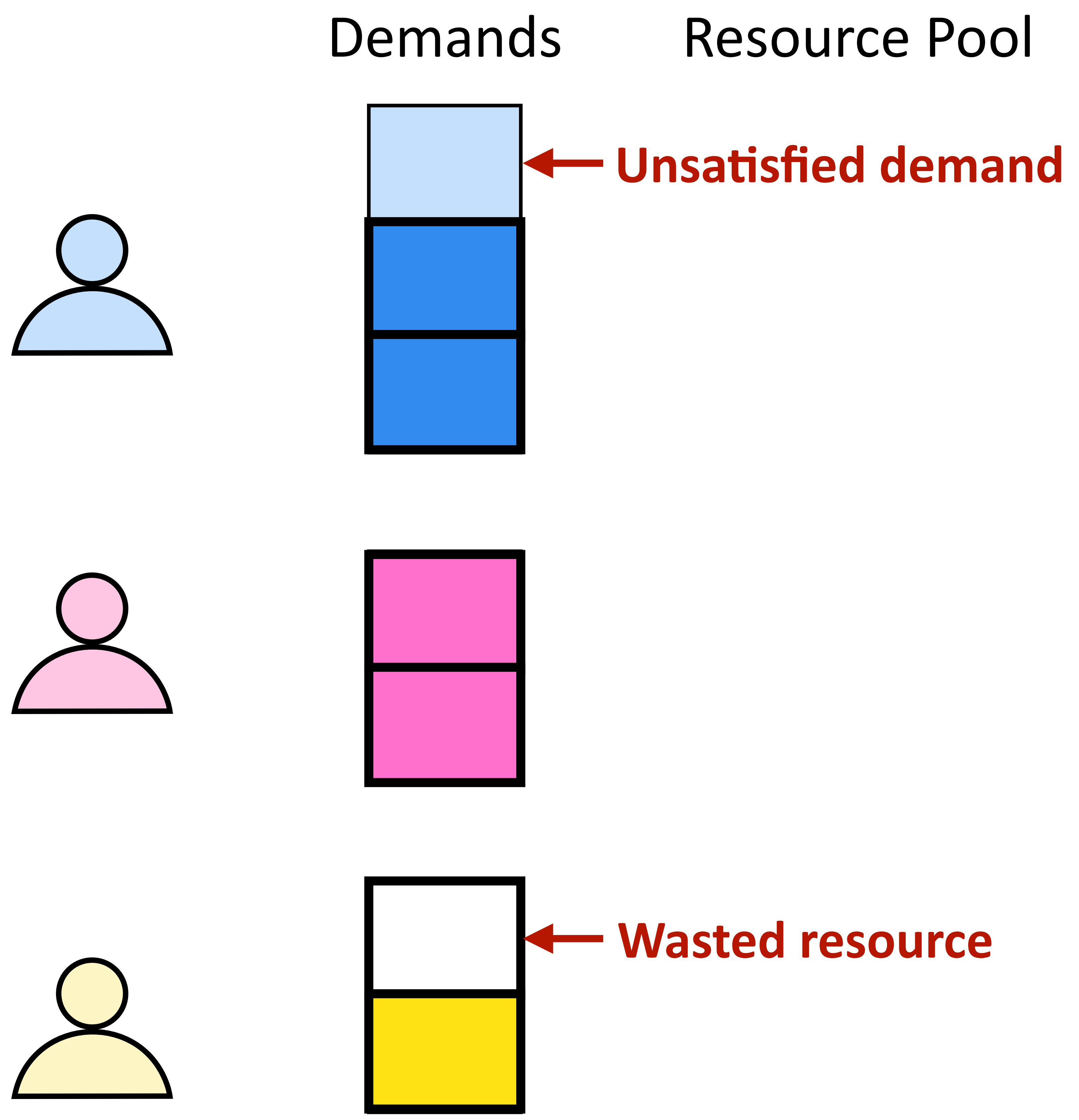
Strict partitioning

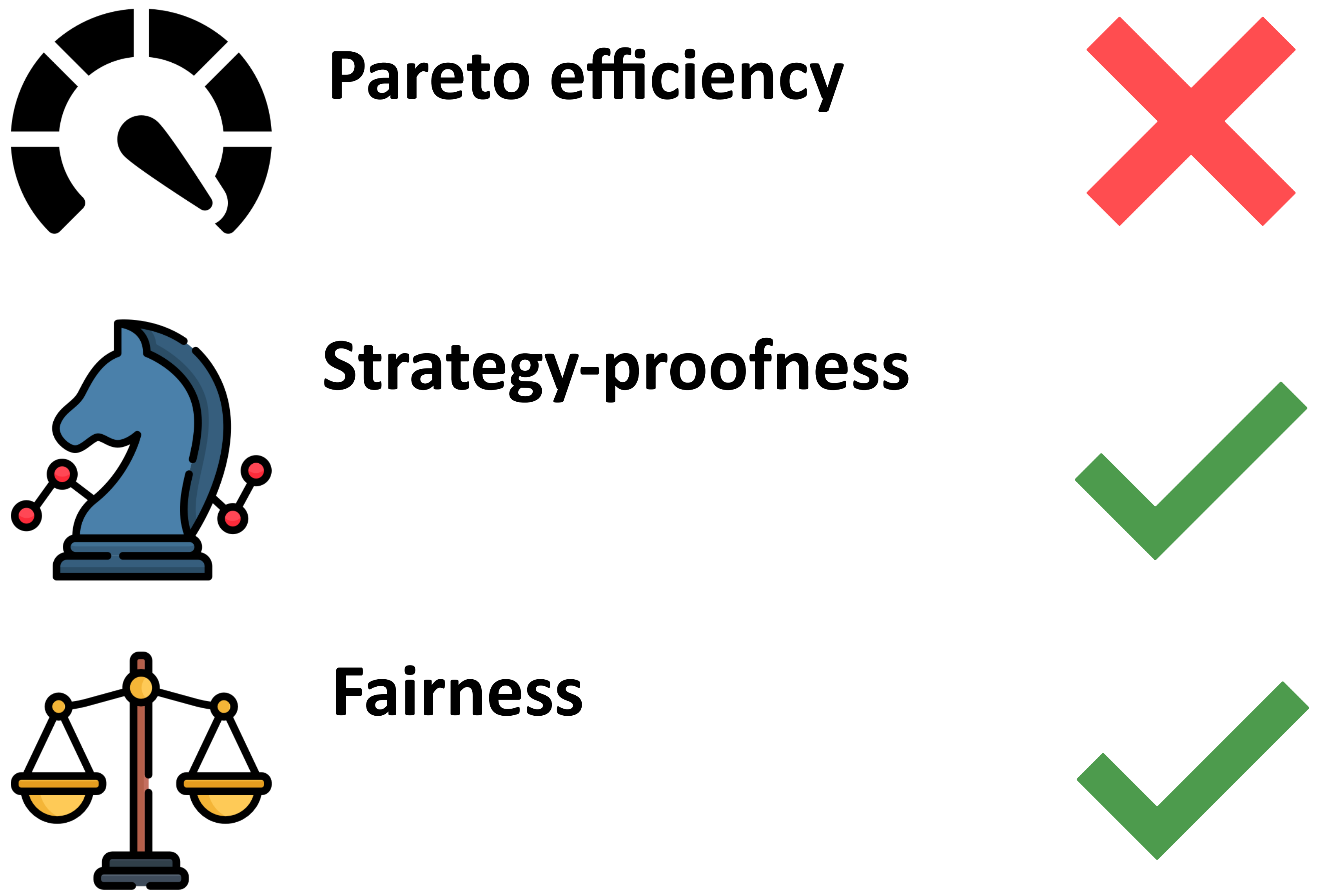
Allocating the resource equally across all users ("fair share") **independent of their demands**



Strict partitioning

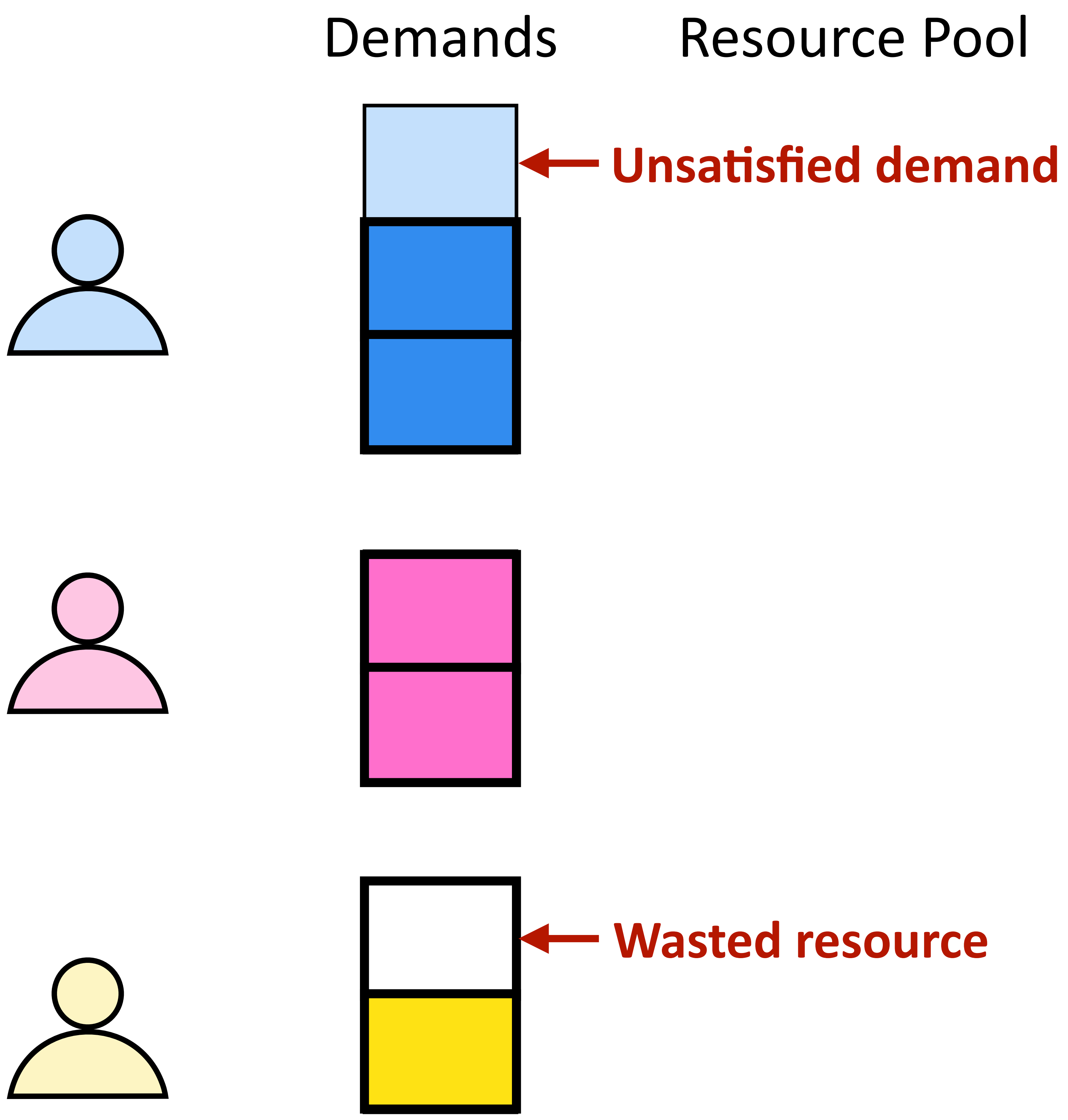
Allocating the resource equally across all users ("fair share") **independent of their demands**





Strict partitioning

Allocating the resource equally across all users ("fair share") **independent of their demands**

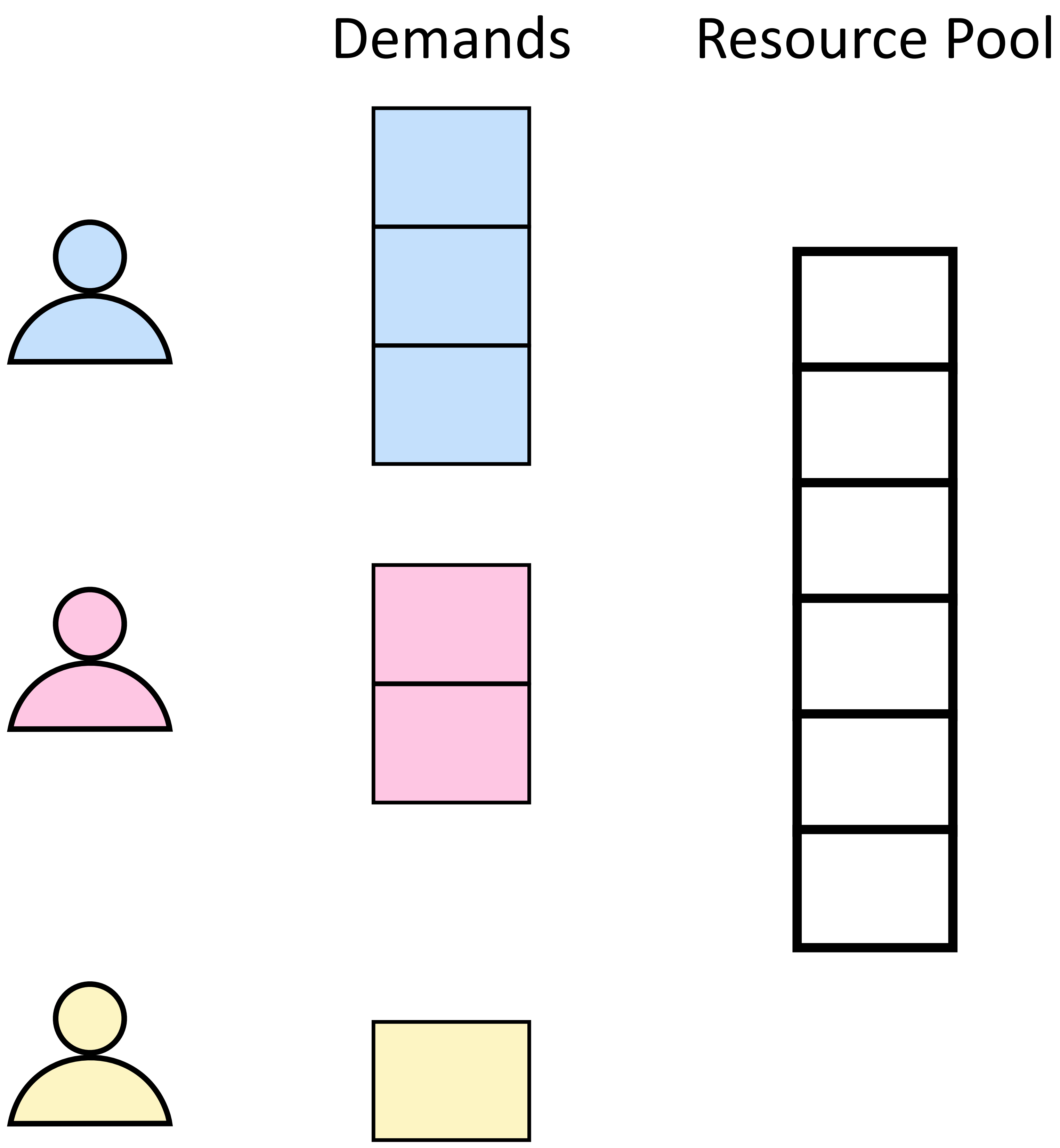


- Pareto efficiency
- Strategy-proofness
- Fairness

Strict partitioning does not guarantee Pareto efficiency

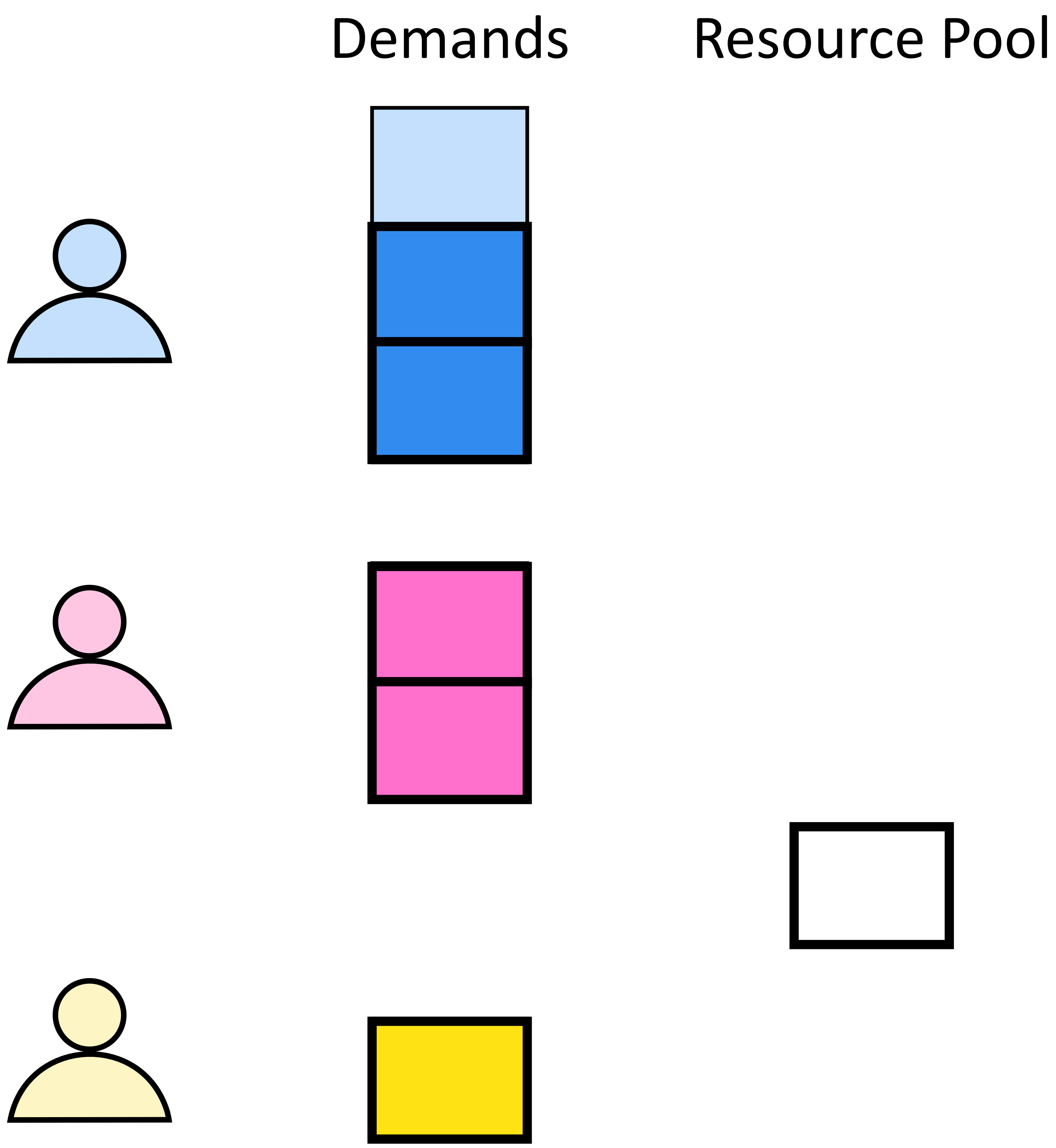
Max-min Fairness alleviates the limitations of strict partitioning (under an assumption)

Maximizing minimum allocation across users while ensuring allocation ≤ demand for each user



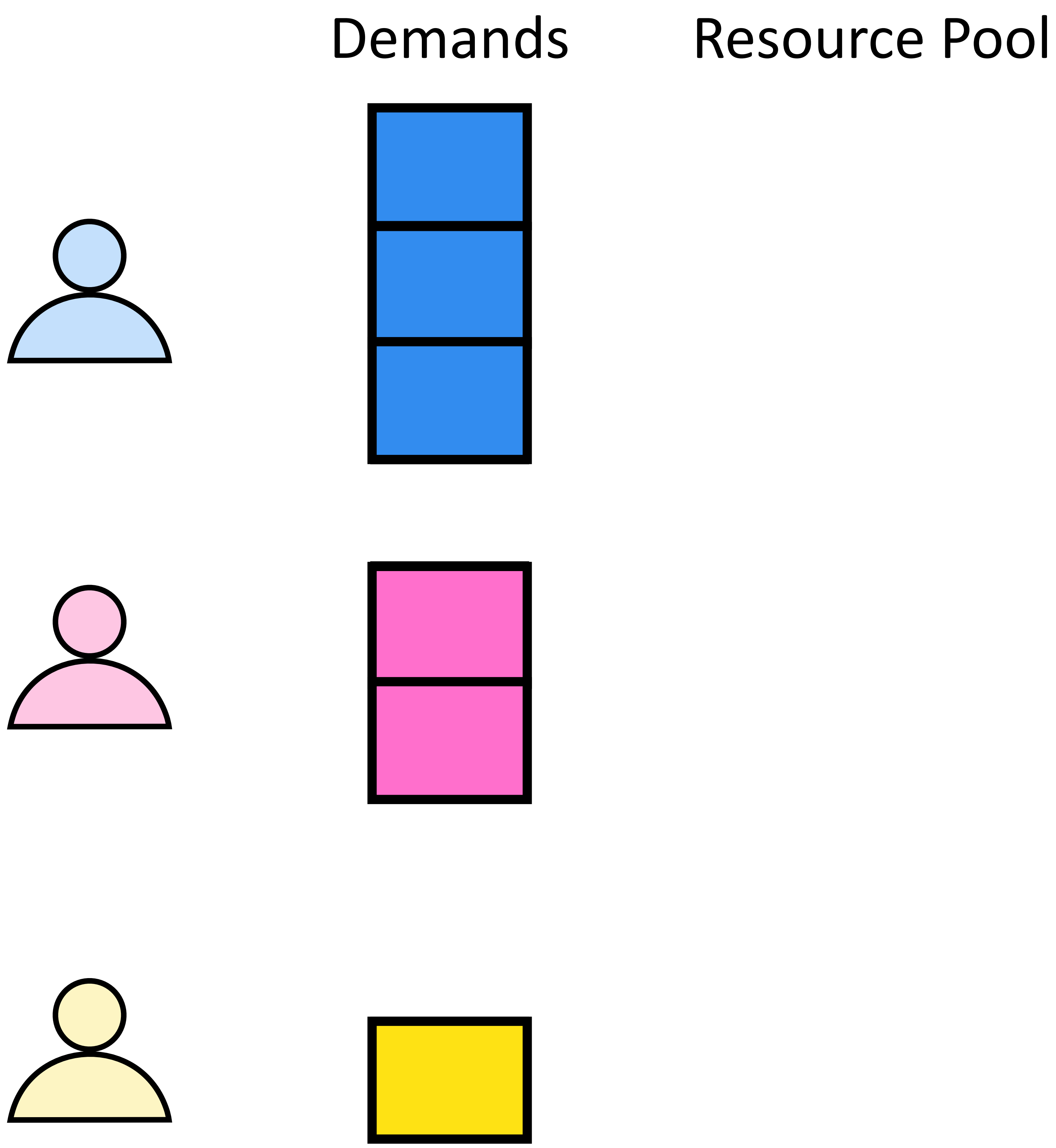
Max-min Fairness alleviates the limitations of strict partitioning (under an assumption)

Maximizing minimum allocation across users while ensuring allocation ≤ demand for each user



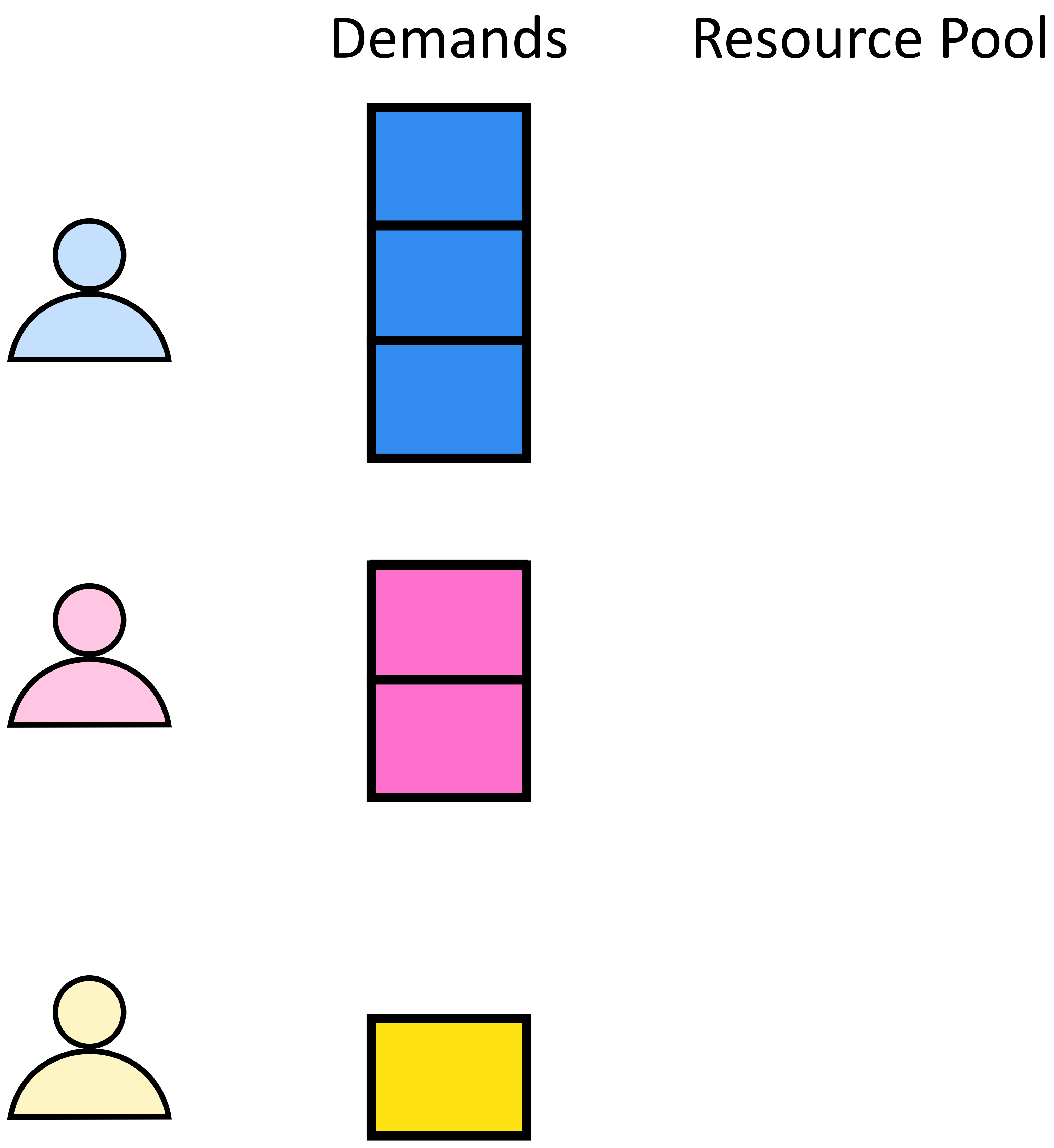
Max-min Fairness alleviates the limitations of strict partitioning (under an assumption)

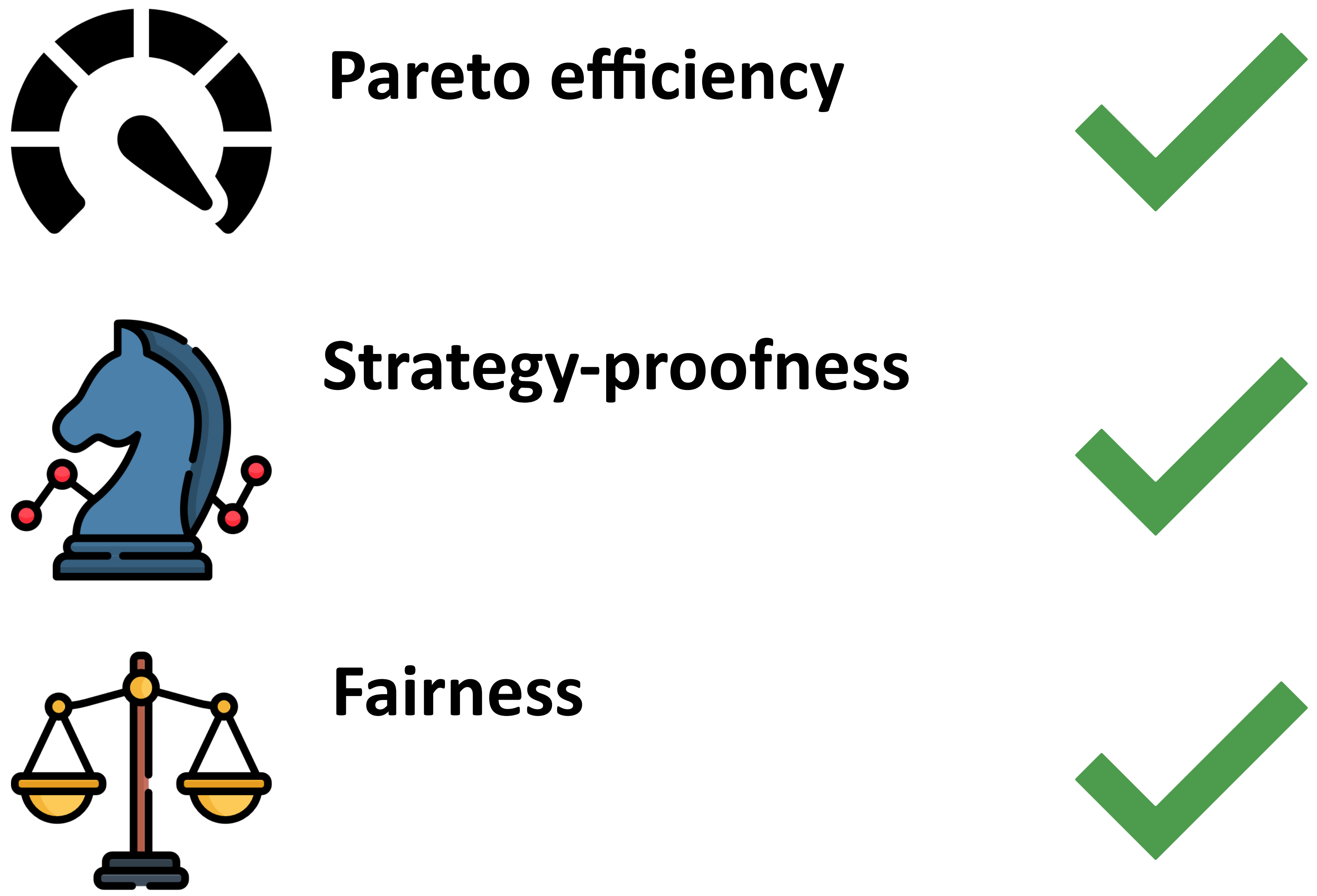
Maximizing minimum allocation across users while ensuring allocation ≤ demand for each user



Max-min Fairness alleviates the limitations of strict partitioning (under an assumption)

Maximizing minimum allocation across users while ensuring allocation ≤ demand for each user





Classical result: Max-min fairness satisfies Pareto efficiency, strategy-proofness and fairness

Underlying assumption: User demands are static

Dynamic demands are the norm in real world deployments

Dynamic demands are the norm in real world deployments

Shared Analytics Clusters

Dynamic demands are the norm in real world deployments

Shared Analytics Clusters

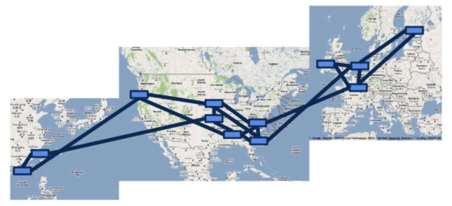
In-memory Key-Value Caches

Dynamic demands are the norm in real world deployments

Shared Analytics Clusters

In-memory Key-Value Caches

Inter-datacenter Network Links



Dynamic demands are the norm in real world deployments

Analysis of real world workloads

Shared Analytics Clusters

In-memory Key-Value Caches

Inter-datacenter Network Links

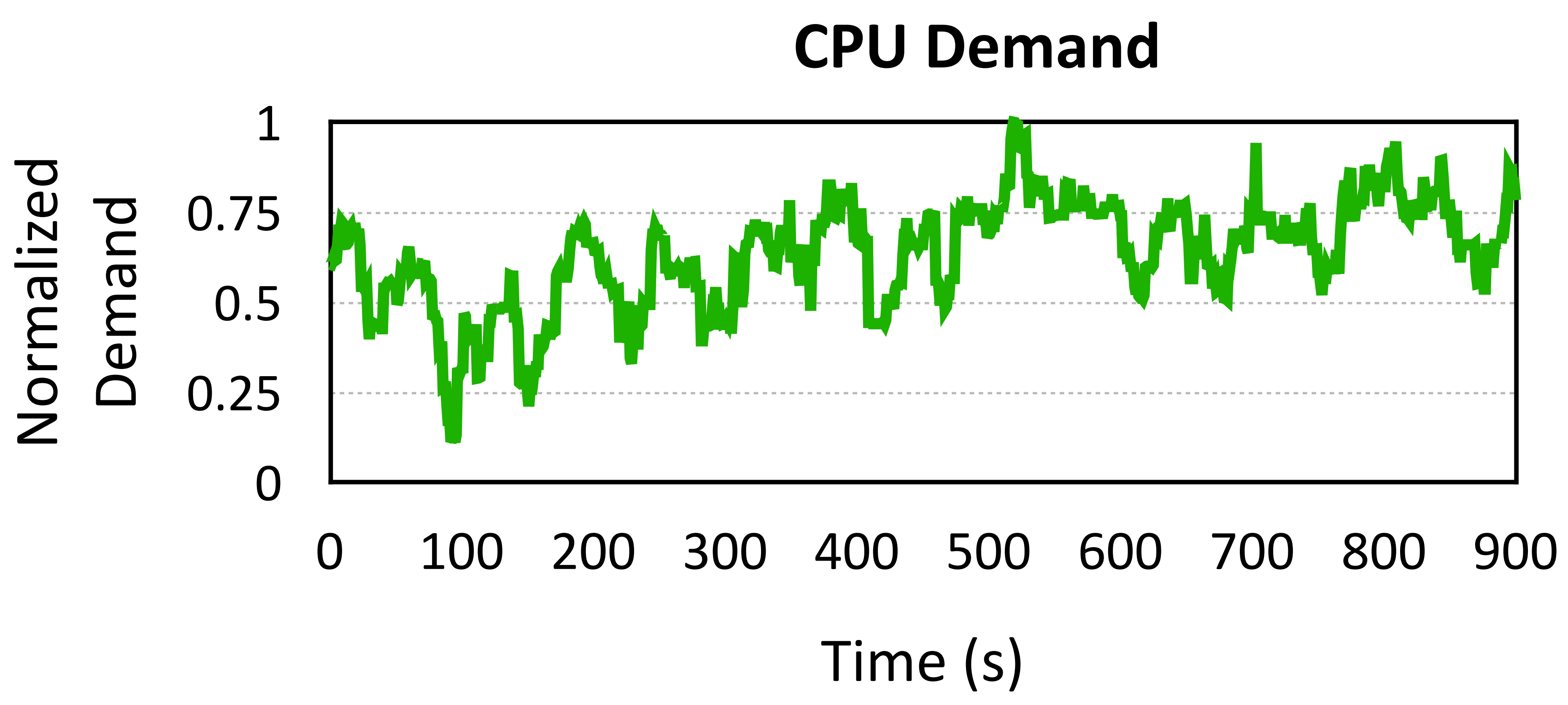
Dynamic demands are the norm in real world deployments

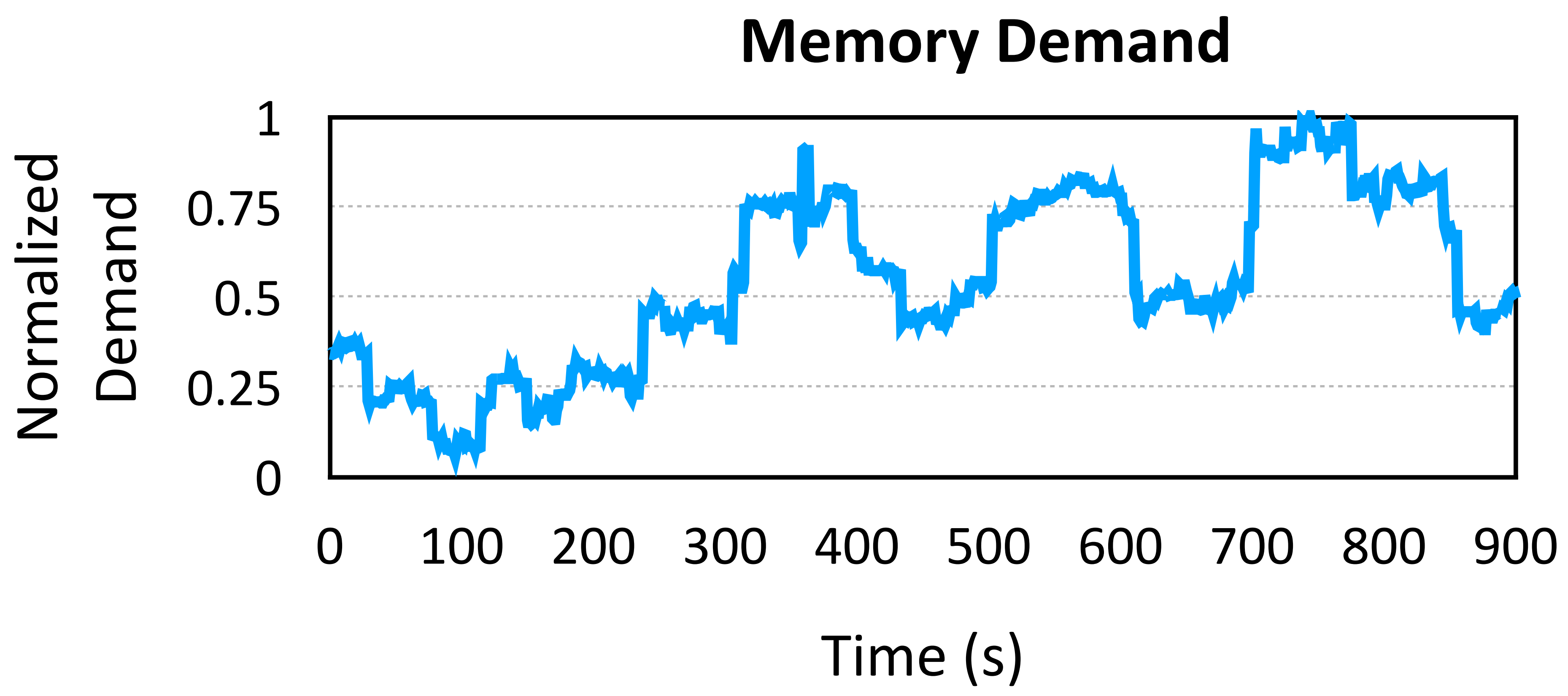
Shared Analytics Clusters

In-memory Key-Value Caches

Inter-datacenter Network Links

Analysis of real world workloads





Dynamic demands are the norm in real world deployments

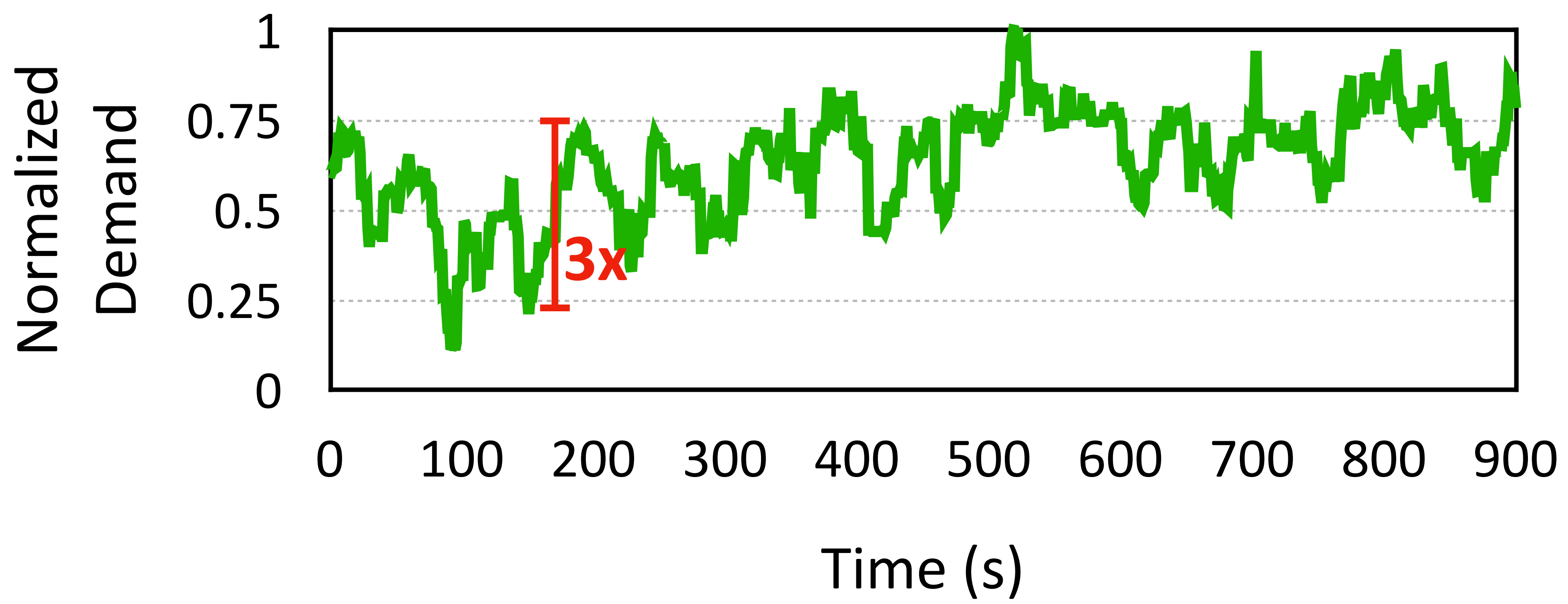
Shared Analytics Clusters

In-memory Key-Value Caches

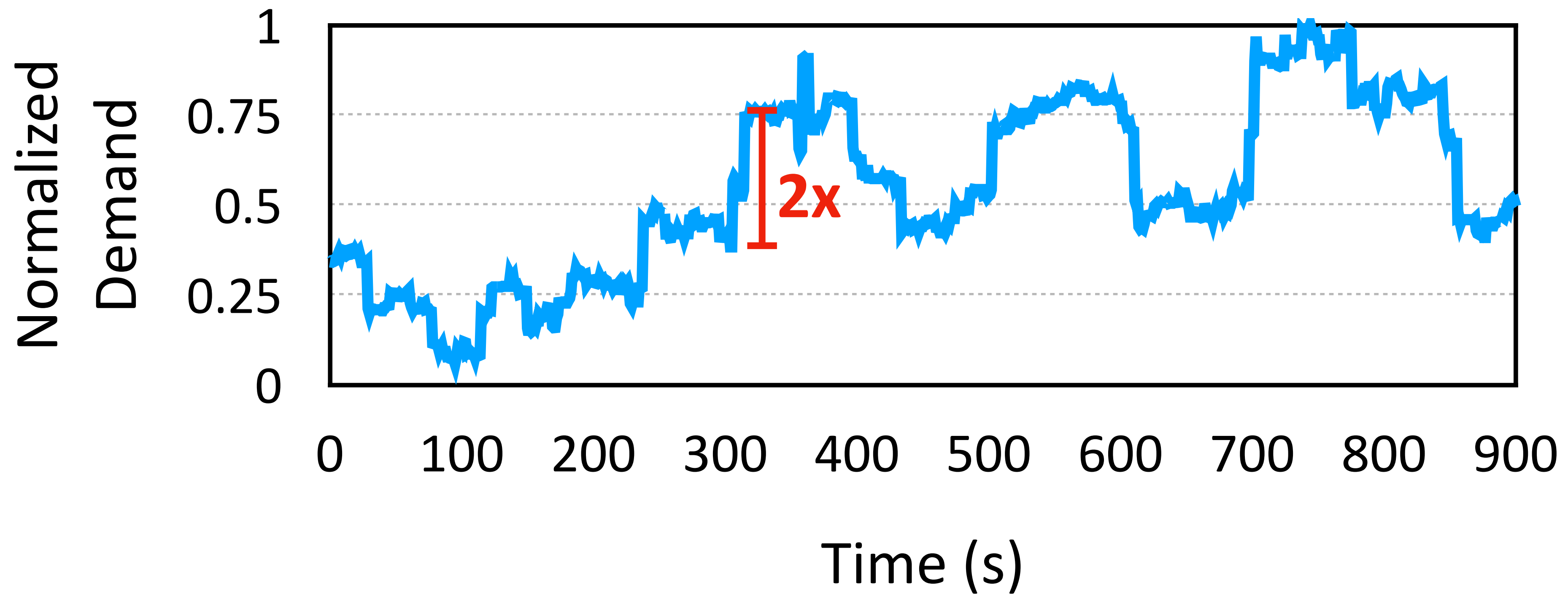
Inter-datacenter Network Links

Analysis of real world workloads

CPU Demand



Memory Demand



2-3x variation in demands over 10s of seconds

Dynamic demands are the norm in real world deployments

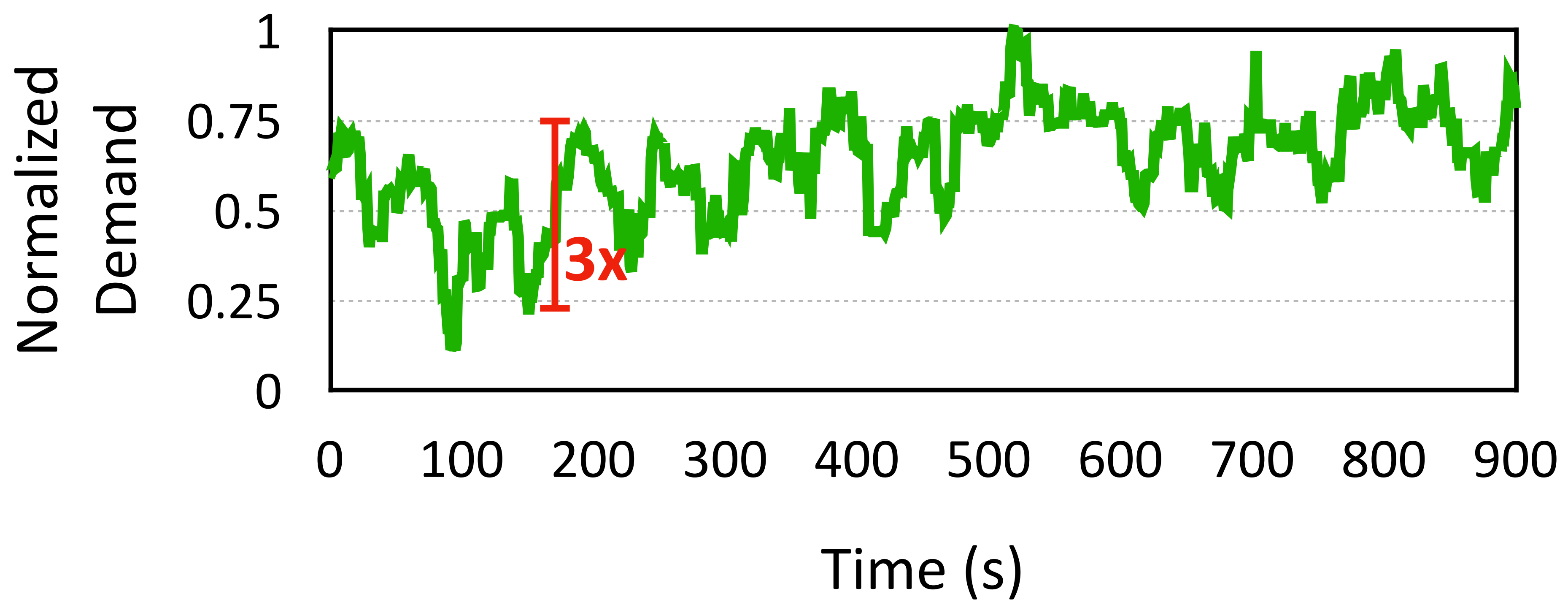
Shared Analytics Clusters

In-memory Key-Value Caches

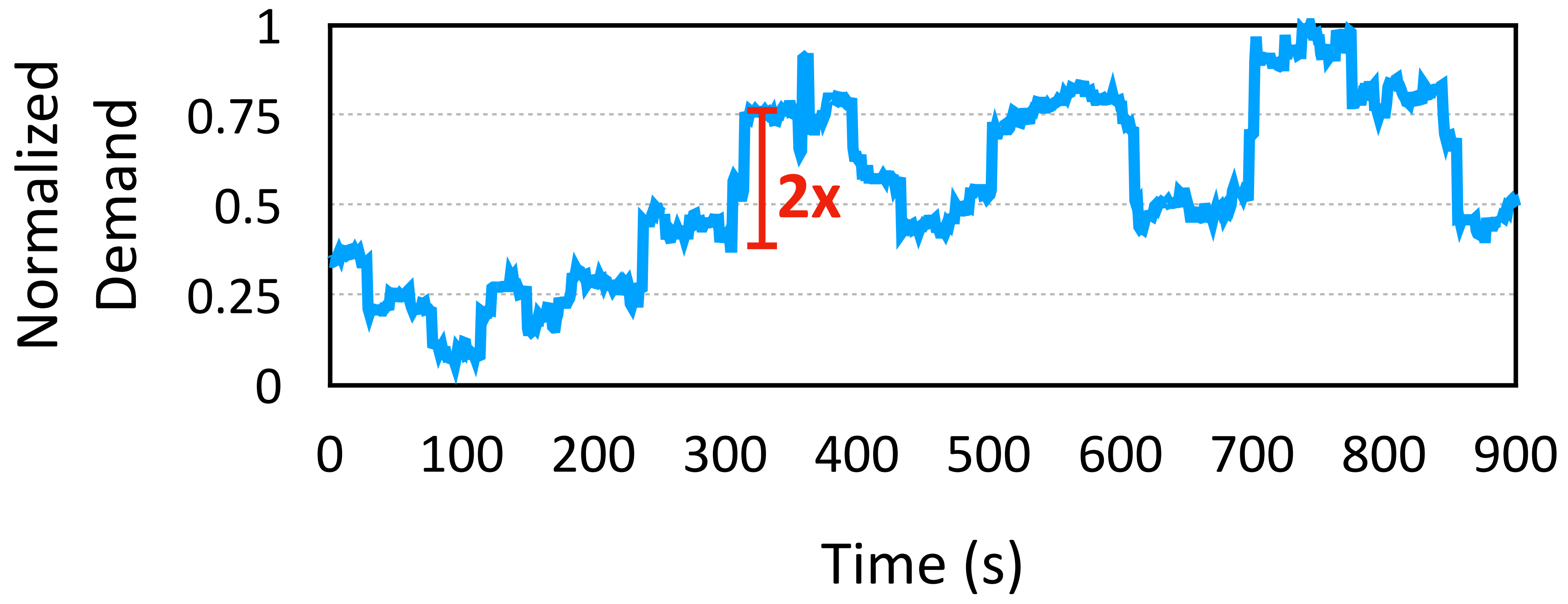
Inter-datacenter Network Links

Analysis of real world workloads

CPU Demand



Memory Demand



2-3x variation in demands over 10s of seconds

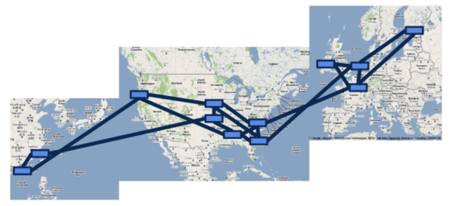
40-70% of users have standard deviation of demands > 1/2 mean

Dynamic demands are the norm in real world deployments

Shared Analytics Clusters

2-3x variation in demands over 10s of seconds

In-memory Key-Value Caches



Inter-datacenter Network Links

Dynamic demands are the norm in real world deployments

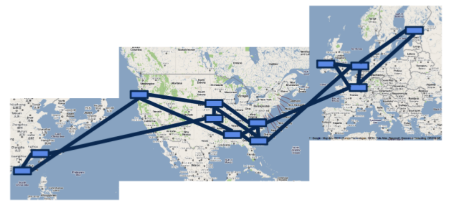
Shared Analytics Clusters

2-3x variation in demands over 10s of seconds

In-memory Key-Value Caches

More than 5x variation in demands within an hour

[Twitter, Yang et al., OSDI’20]



Inter-datacenter Network Links

35% variation in demands over 5 minute intervals

[Microsoft, Abuzaid et al., NSDI’21]

Dynamic demands are the norm in real world deployments

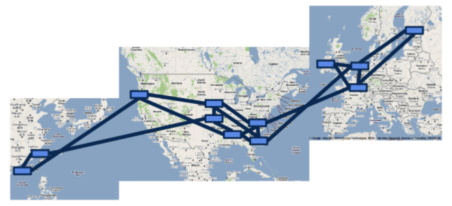
Shared Analytics Clusters

2-3x variation in demands over 10s of seconds

In-memory Key-Value Caches

More than 5x variation in demands within an hour

[Twitter, Yang et al., OSDI'20]



Inter-datacenter Network Links

35% variation in demands over 5 minute intervals

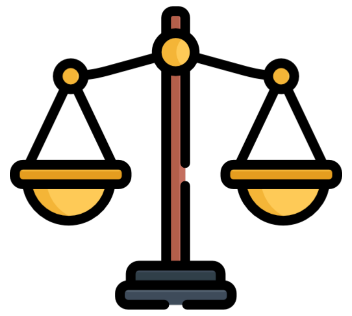
[Microsoft, Abuzaid et al., NSDI'21]

Significant variation in user demands over time in real workloads

Max-min fairness under dynamic demands: looses one or more of its properties


Pareto efficiency


Strategy-proofness

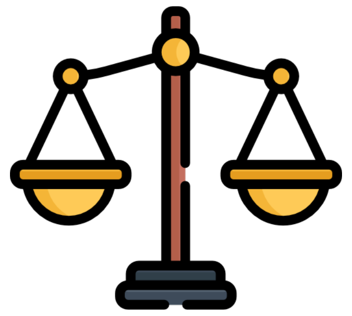

Fairness

Max-min fairness under dynamic demands: looses one or more of its properties

Max-min fairness
based on demands at t=0

Pareto efficiency

Strategy-proofness



Fairness

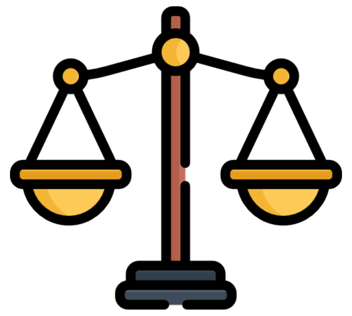
Max-min fairness under dynamic demands: looses one or more of its properties

Max-min fairness
based on demands at t=0


Pareto efficiency



Strategy-proofness



Fairness

(Explained in paper)

Max-min fairness under dynamic demands: looses one or more of its properties

	Max-min fairness based on demands at t=0	Max-min fairness applied periodically
Pareto efficiency	✗	
Strategy-proofness	✗	
Fairness		

(Explained in paper)

Max-min fairness under dynamic demands: looses one or more of its properties

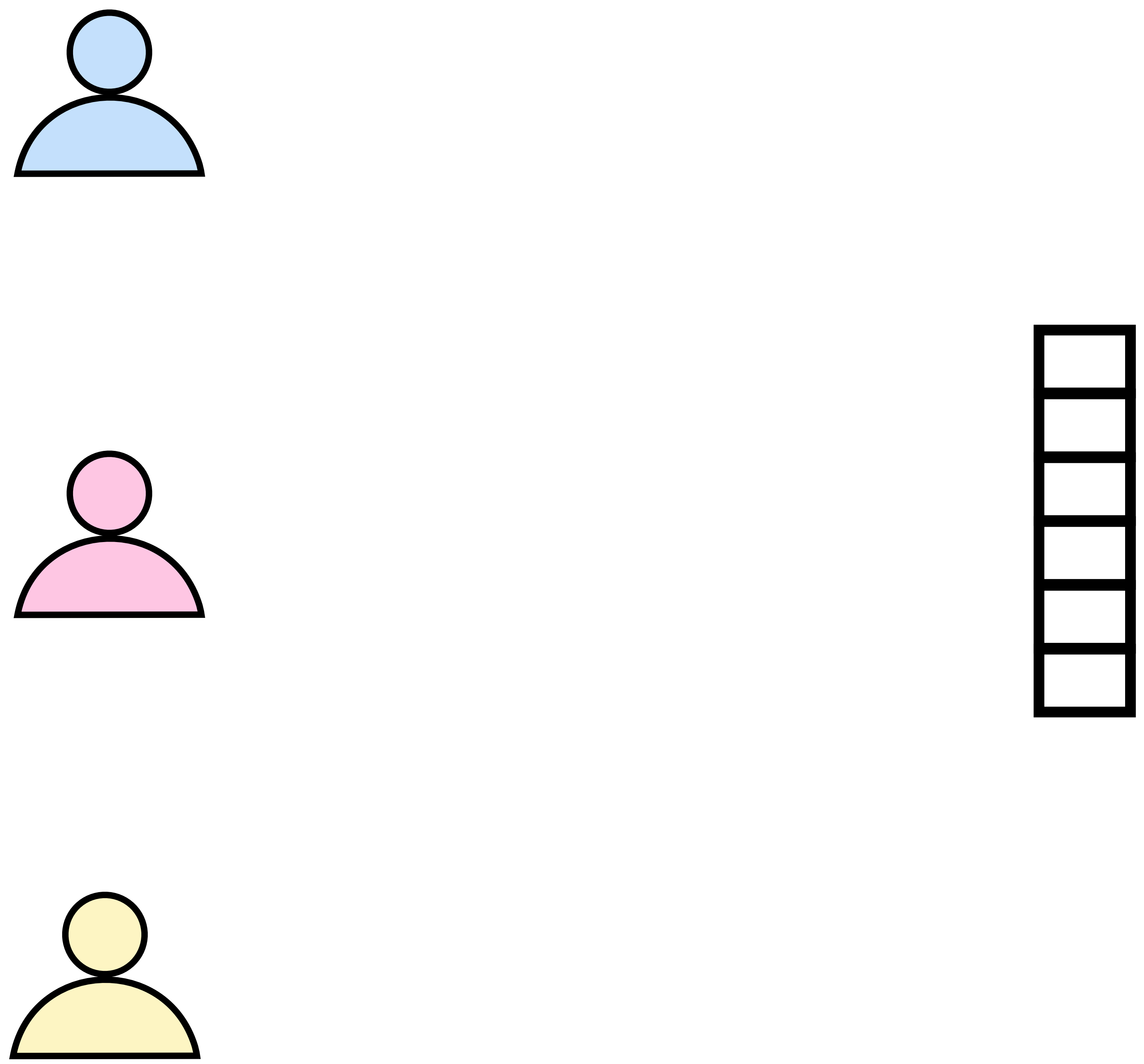
		Max-min fairness based on demands at t=0	Max-min fairness applied periodically
	Pareto efficiency	✗	
	Strategy-proofness	✗	
	Fairness		

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

Demands

Resource Pool



Max-min fairness based on demands at t=0

Max-min fairness applied periodically

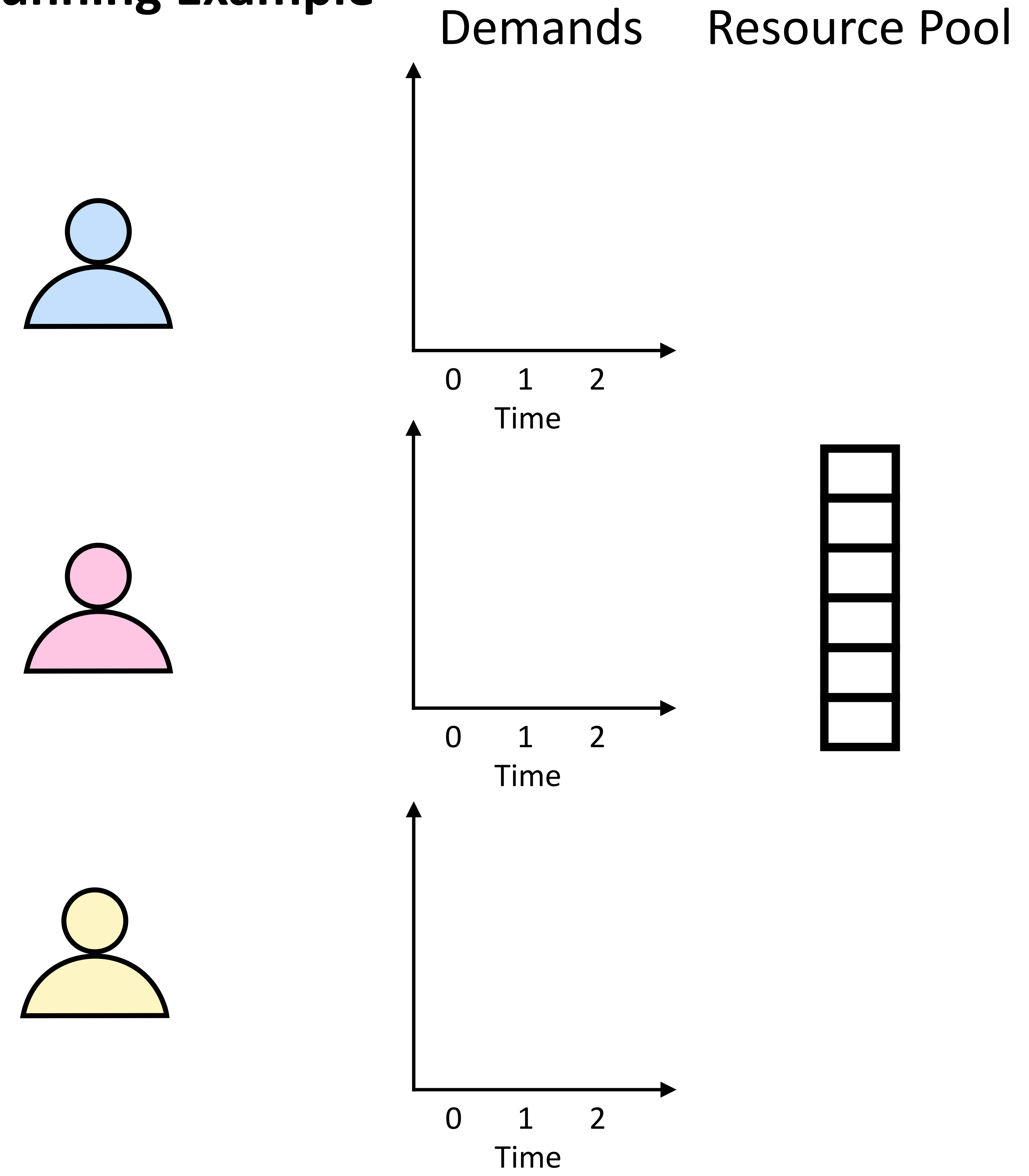
Pareto efficiency ✗

Strategy-proofness ✗

Fairness

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example



Max-min fairness based on demands at t=0

Max-min fairness applied periodically

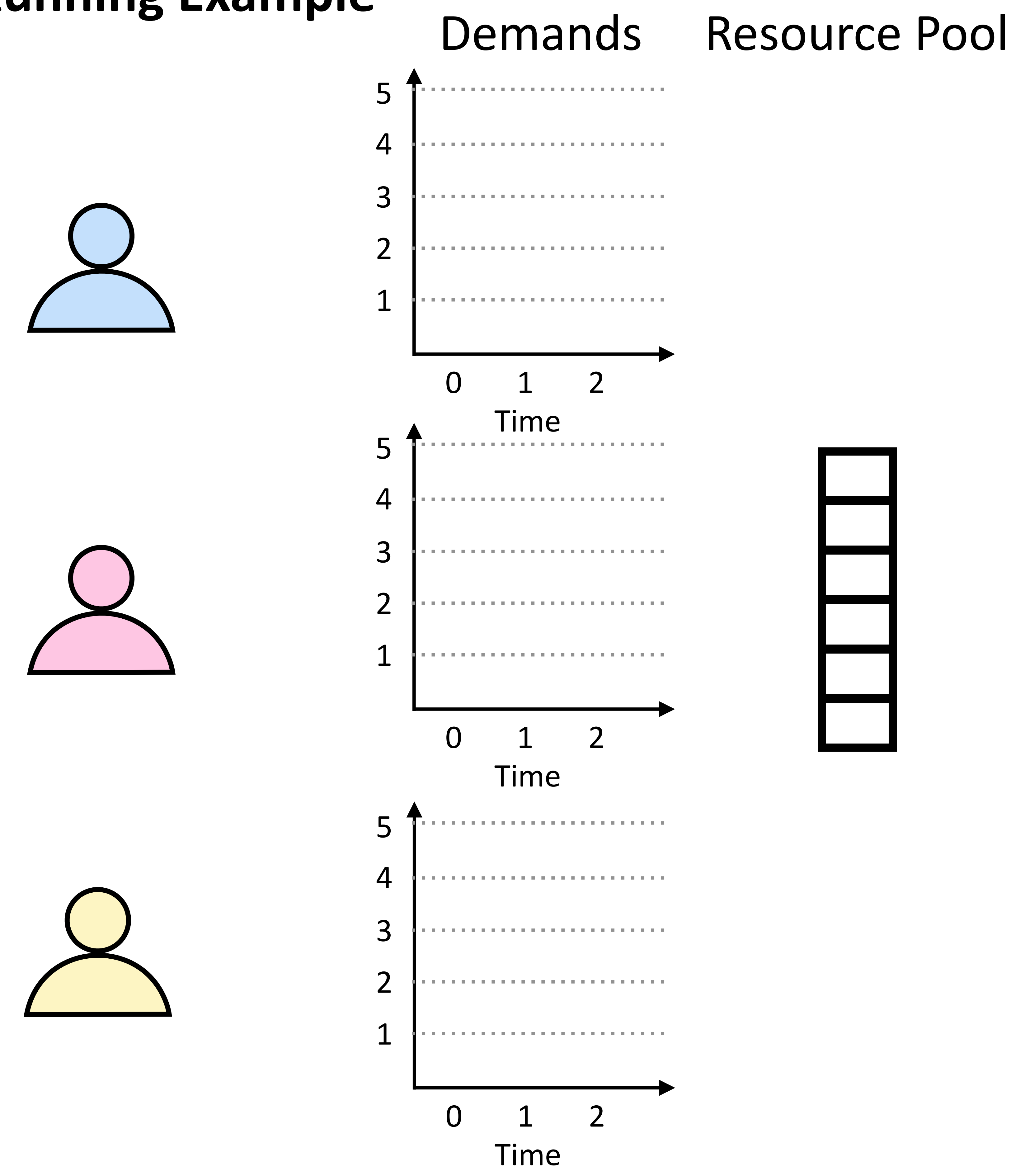
Pareto efficiency

Strategy-proofness

Fairness

Max-min fairness under dynamic demands: looses one or more of its properties

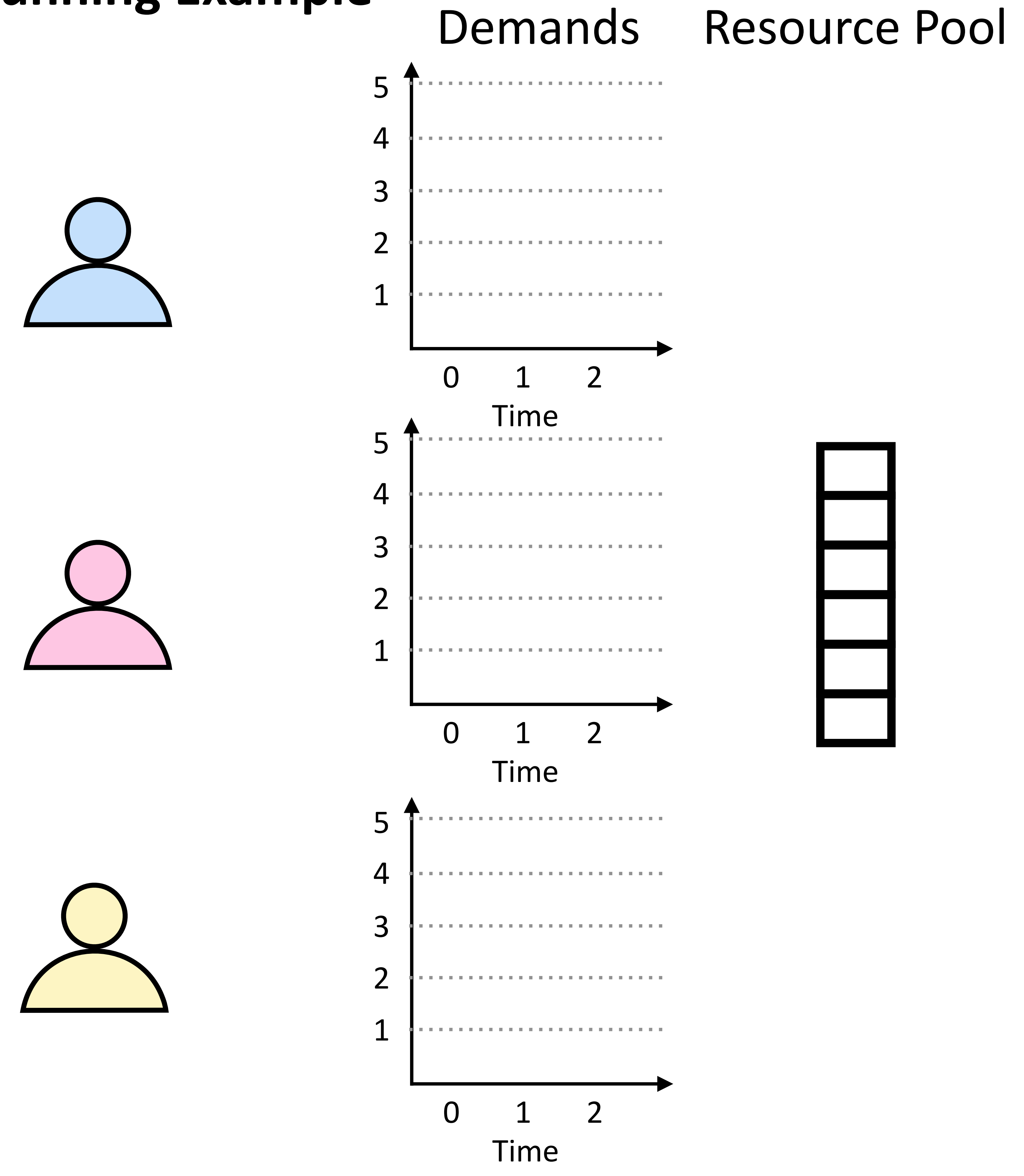
Running Example



	Max-min fairness based on demands at t=0	Max-min fairness applied periodically
Pareto efficiency	✗	
Strategy-proofness	✗	
Fairness		

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

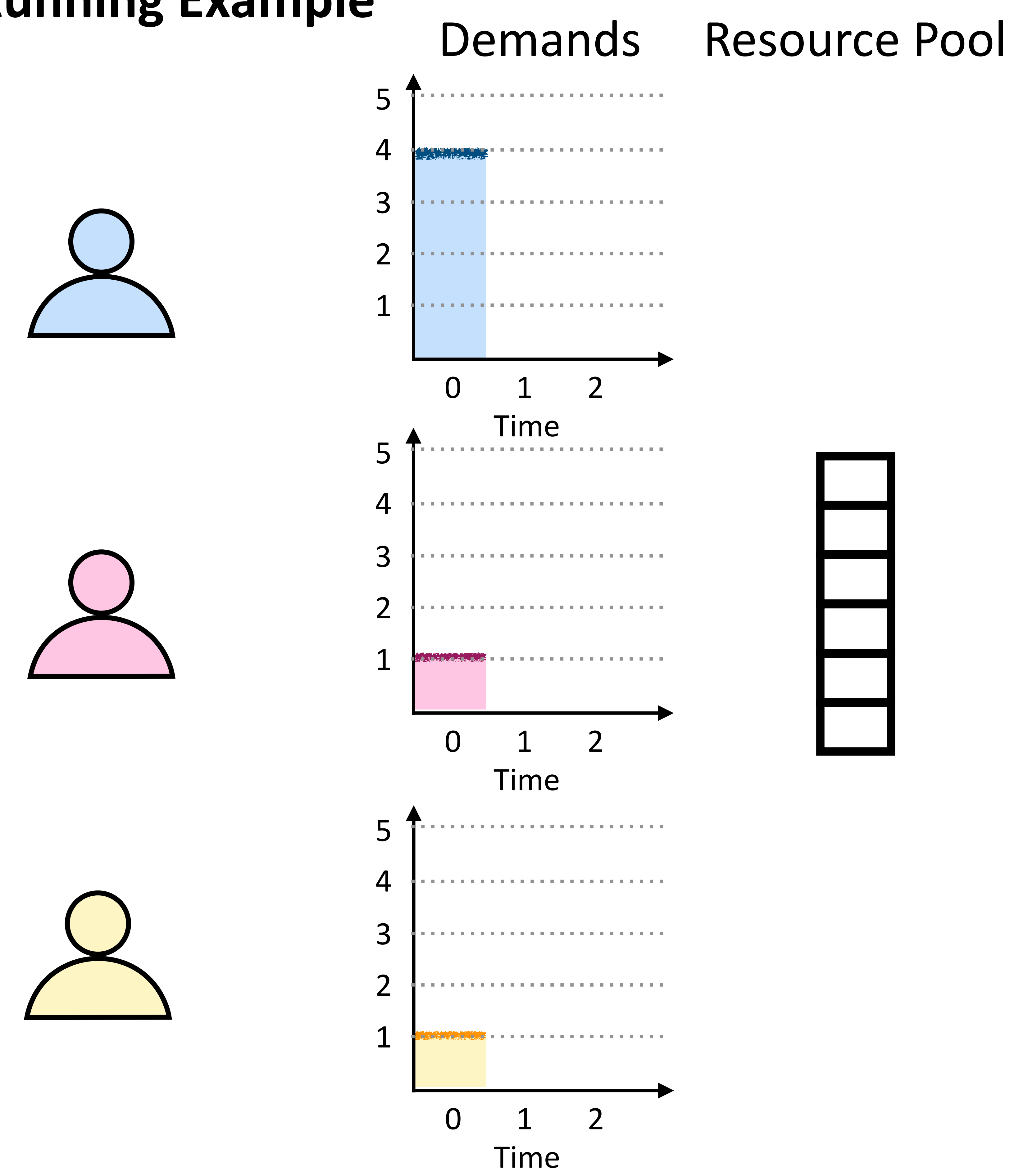


Max-min fairness based on demands at t=0

Max-min fairness applied periodically

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example



Max-min fairness based on demands at t=0

Max-min fairness applied periodically

Pareto efficiency ✗

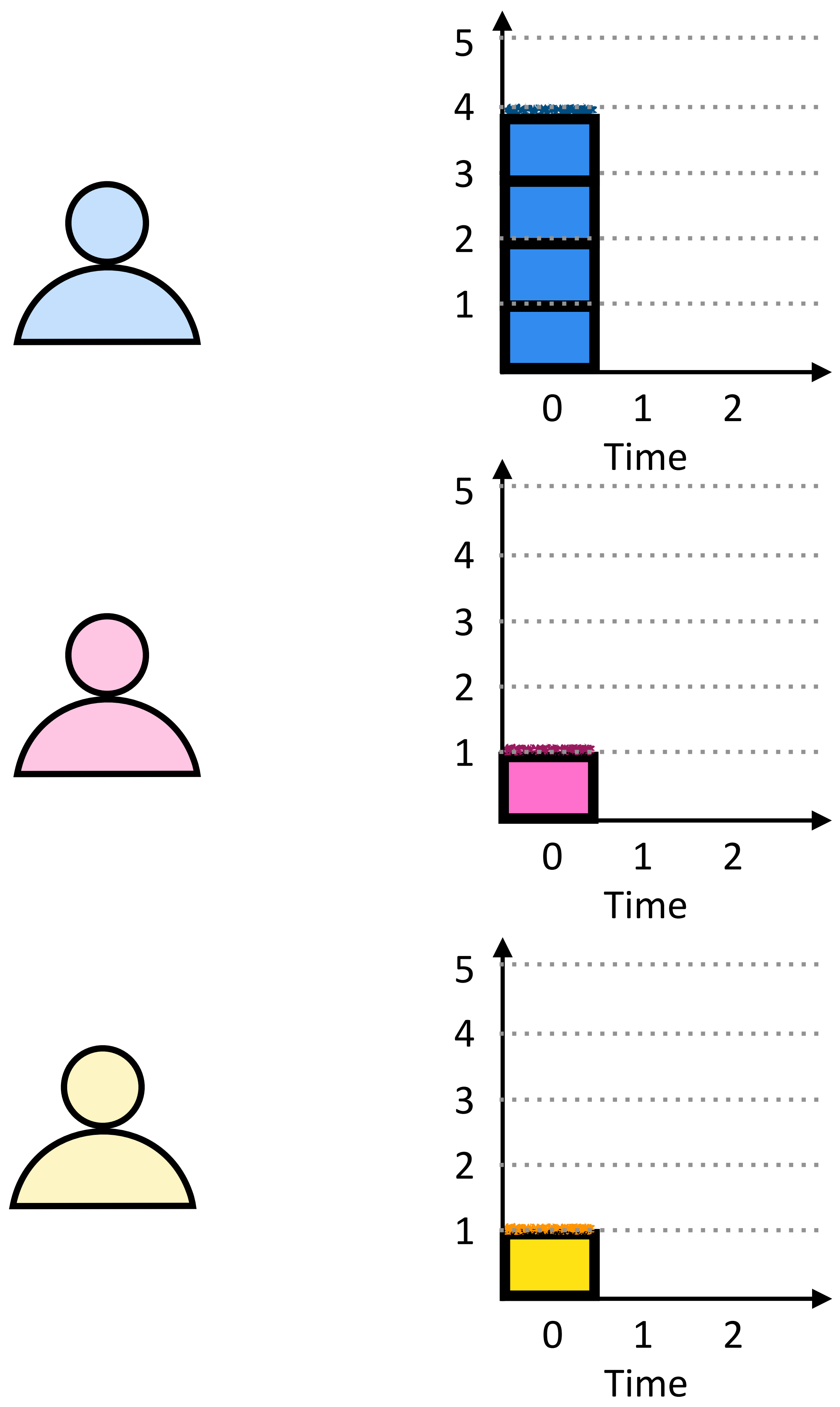
Strategy-proofness ✗

Fairness

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

Demands Resource Pool



Max-min fairness based on demands at t=0

Max-min fairness applied periodically

Pareto efficiency ✗

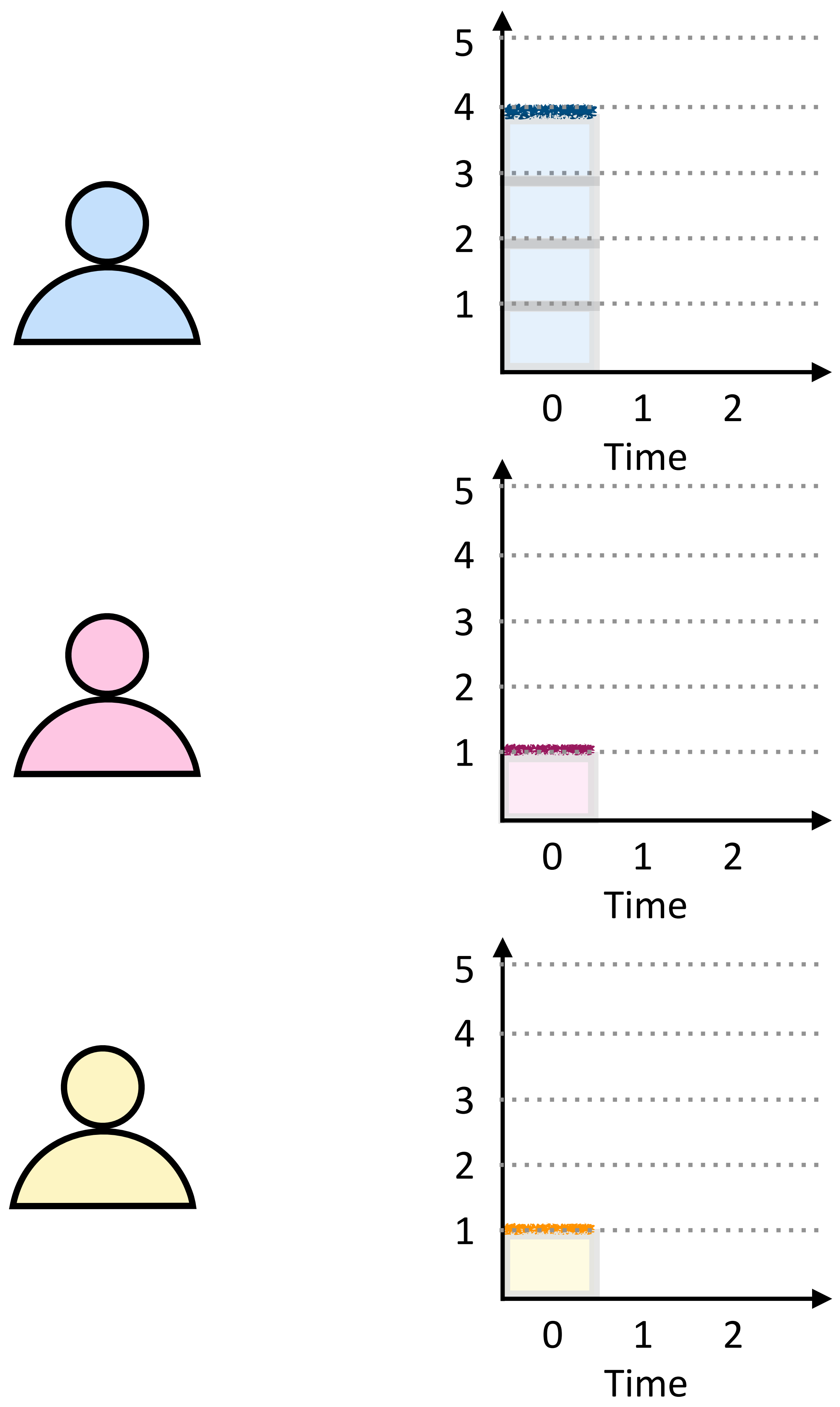
Strategy-proofness ✗

Fairness

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

Demands | Resource Pool

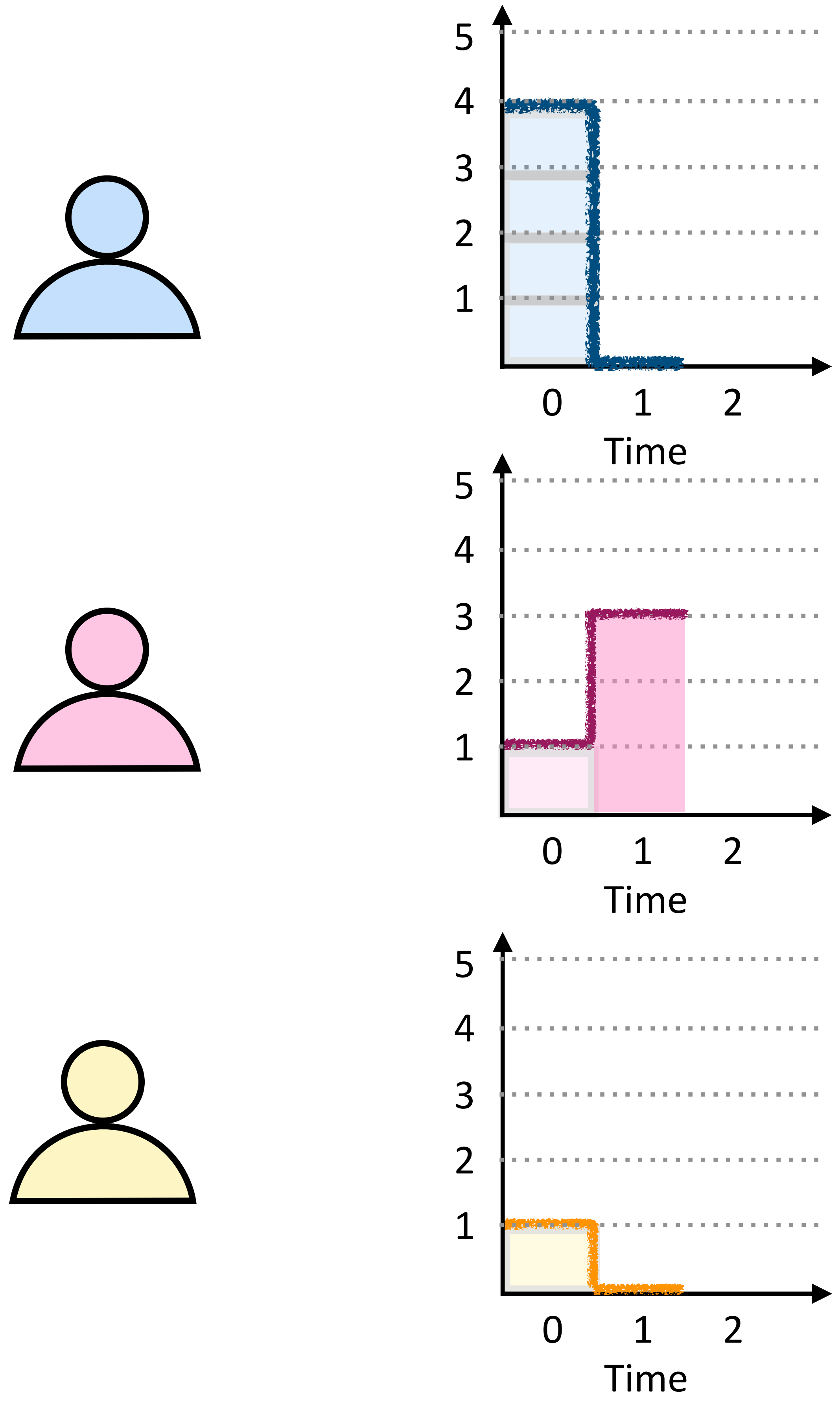


	Max-min fairness based on demands at t=0	Max-min fairness applied periodically
Pareto efficiency	✗	
Strategy-proofness	✗	
Fairness		

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

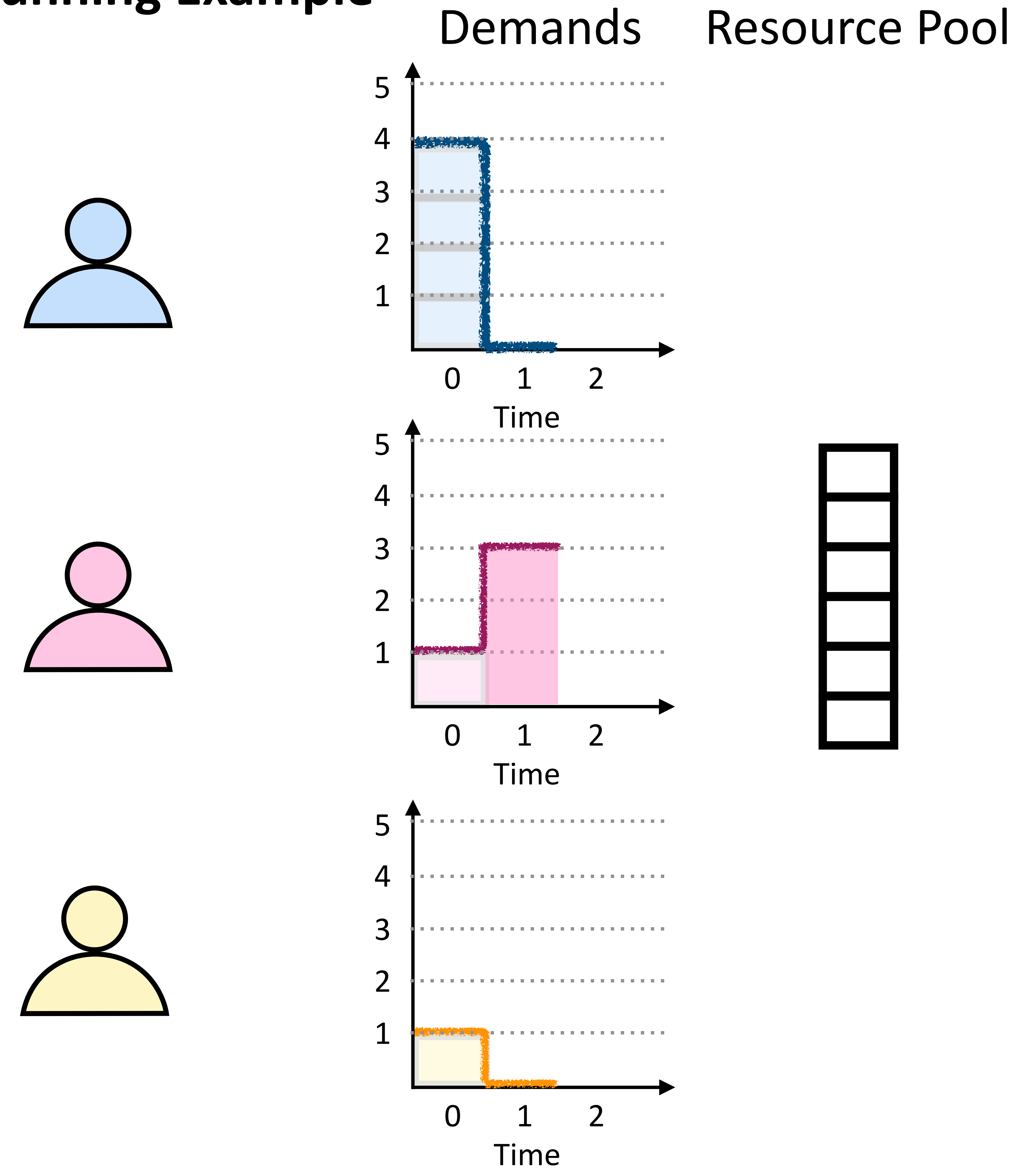
Demands   Resource Pool



	Max-min fairness based on demands at t=0	**Max-min fairness applied periodically**
Pareto efficiency	✗	
Strategy-proofness	✗	
Fairness		

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example



Max-min fairness based on demands at t=0

Max-min fairness applied periodically

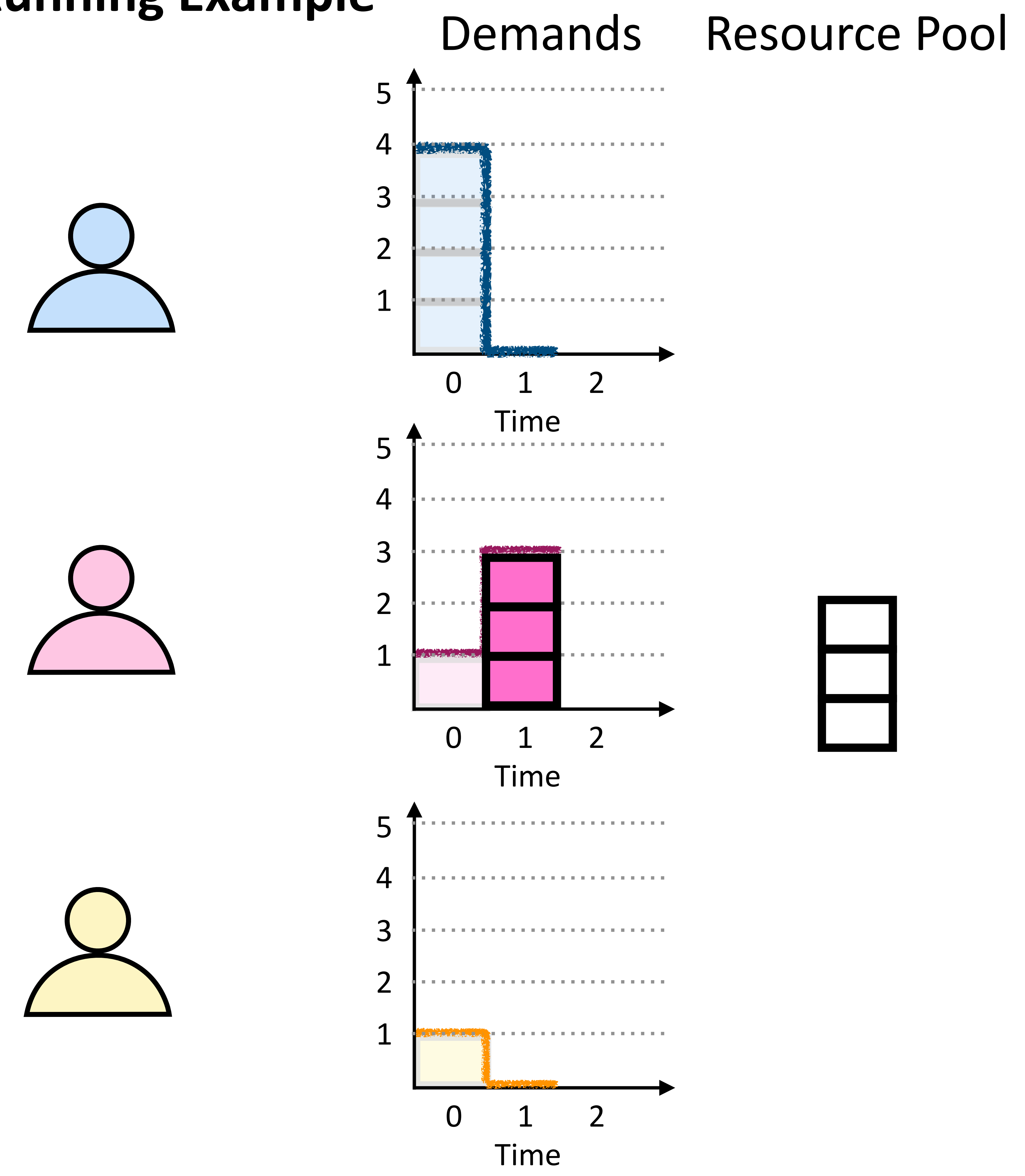
Pareto efficiency ✗

Strategy-proofness ✗

Fairness

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

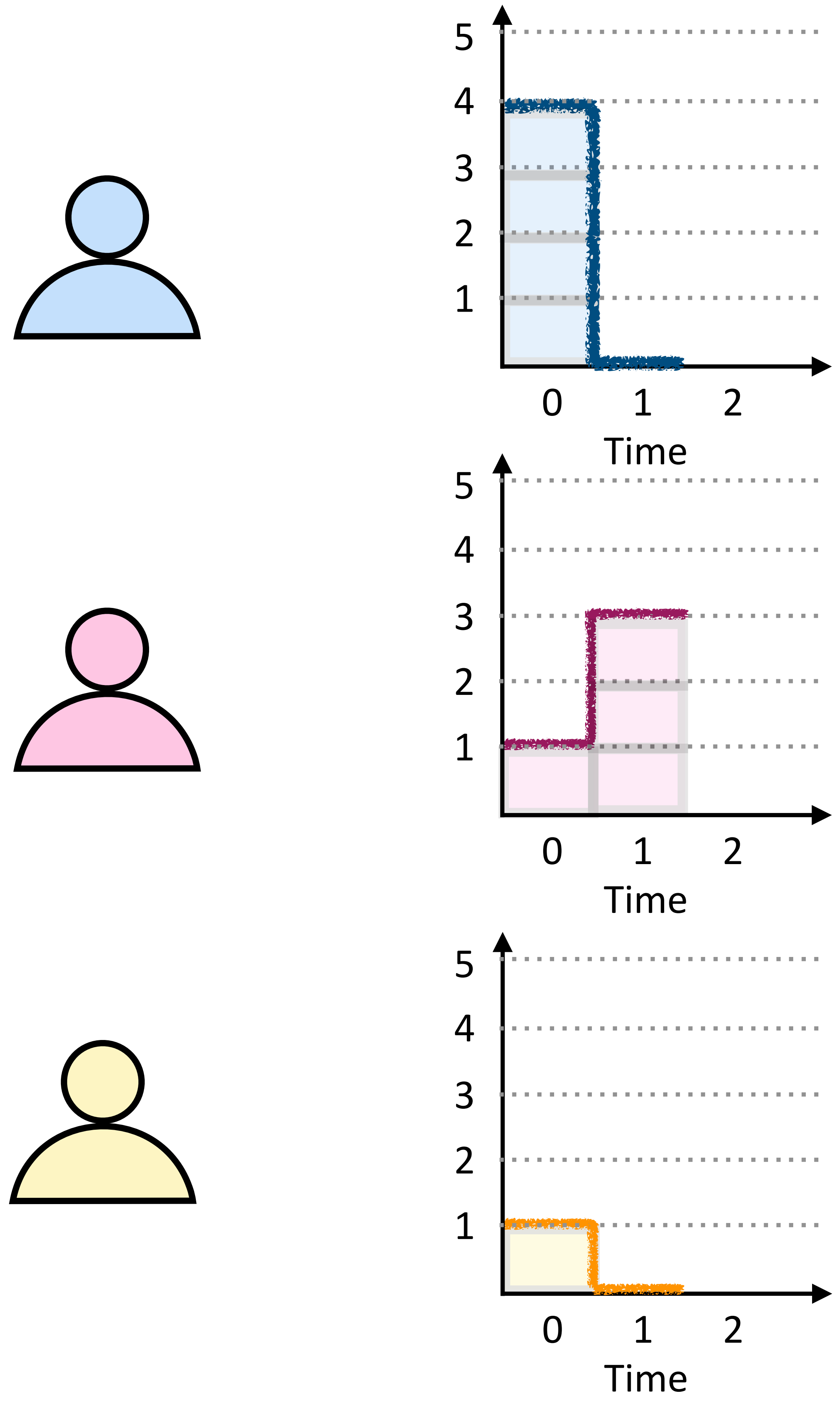


	Max-min fairness based on demands at t=0	Max-min fairness applied periodically
Pareto efficiency	✗	
Strategy-proofness	✗	
Fairness		

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

Demands Resource Pool



Max-min fairness based on demands at t=0

Max-min fairness applied periodically

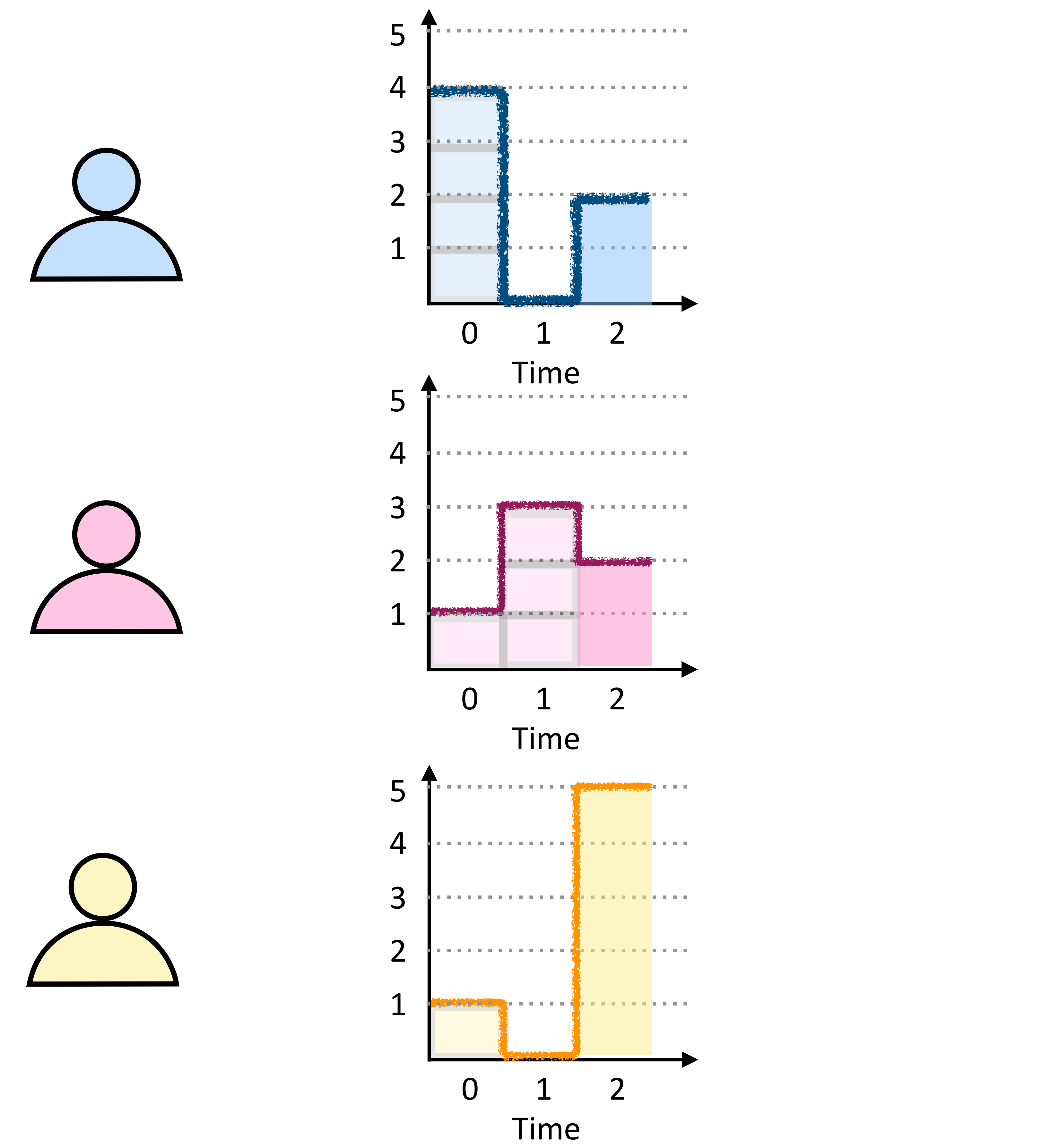
Pareto efficiency

Strategy-proofness

Fairness

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

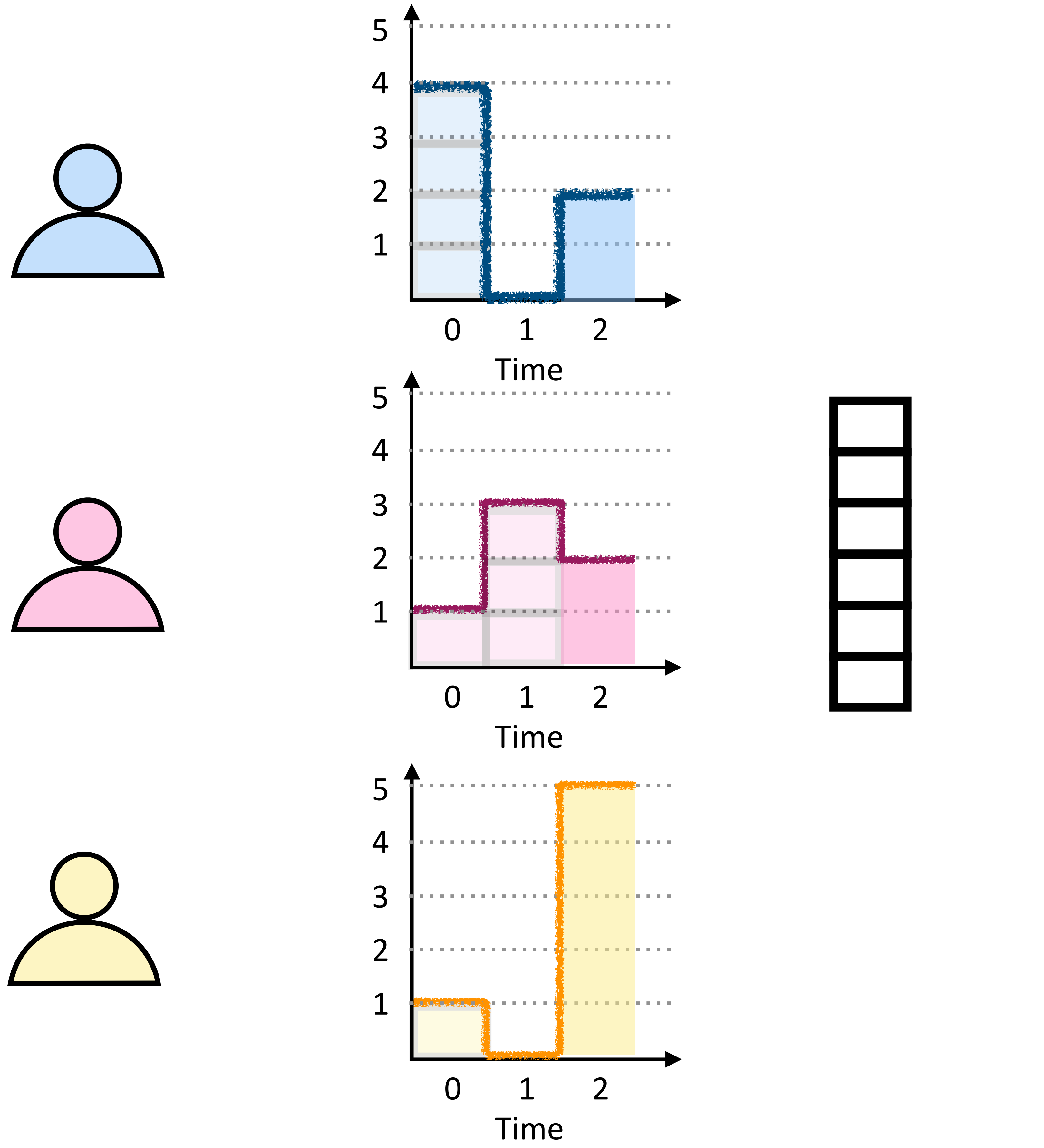


Max-min fairness based on demands at t=0

Max-min fairness applied periodically

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example



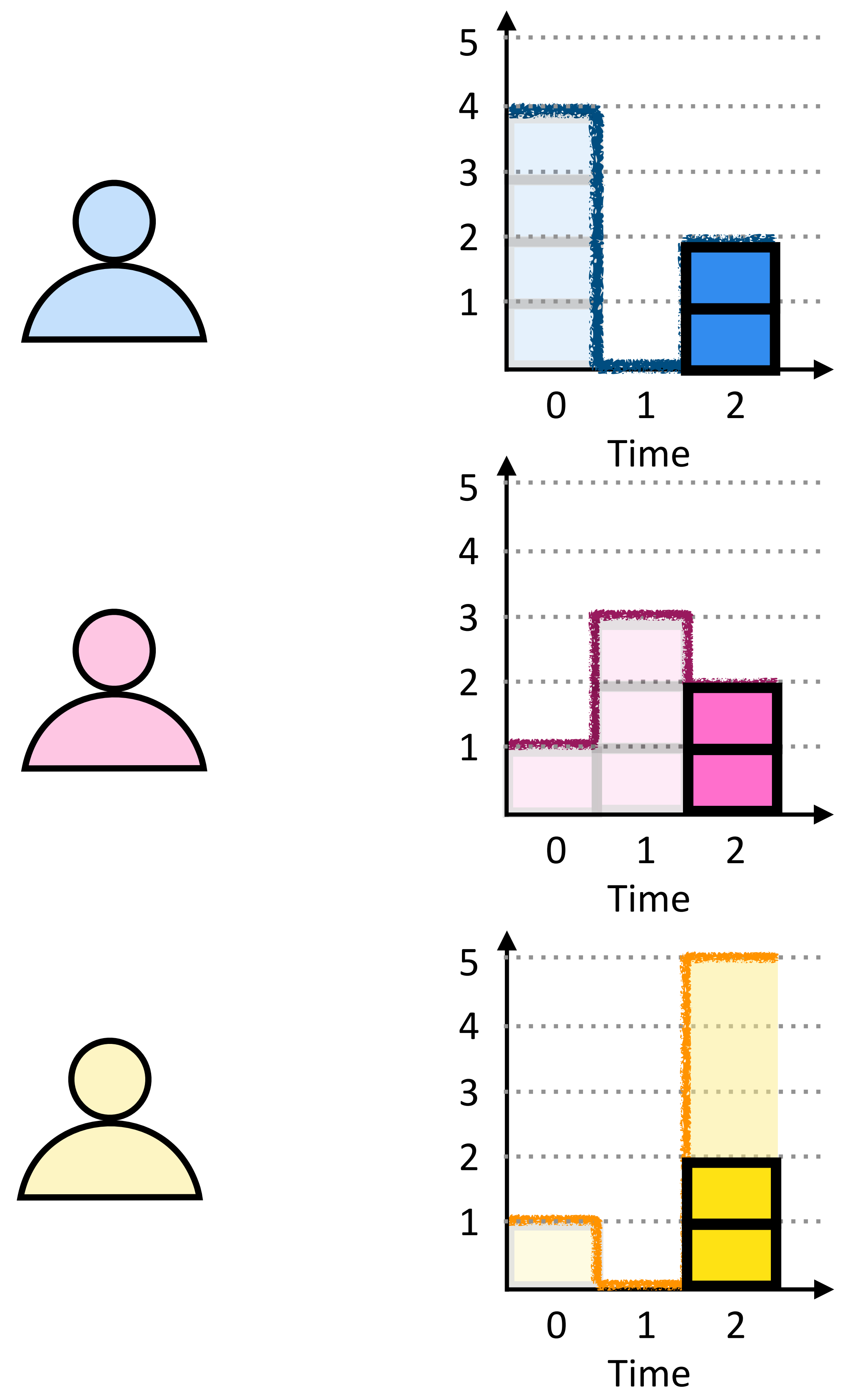
	Max-min fairness based on demands at t=0	**Max-min fairness applied periodically**
Pareto efficiency	✗	
Strategy-proofness	✗	
Fairness		

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

Demands

Resource Pool



Max-min fairness based on demands at t=0

Max-min fairness applied periodically

Pareto efficiency ✗

Strategy-proofness ✗

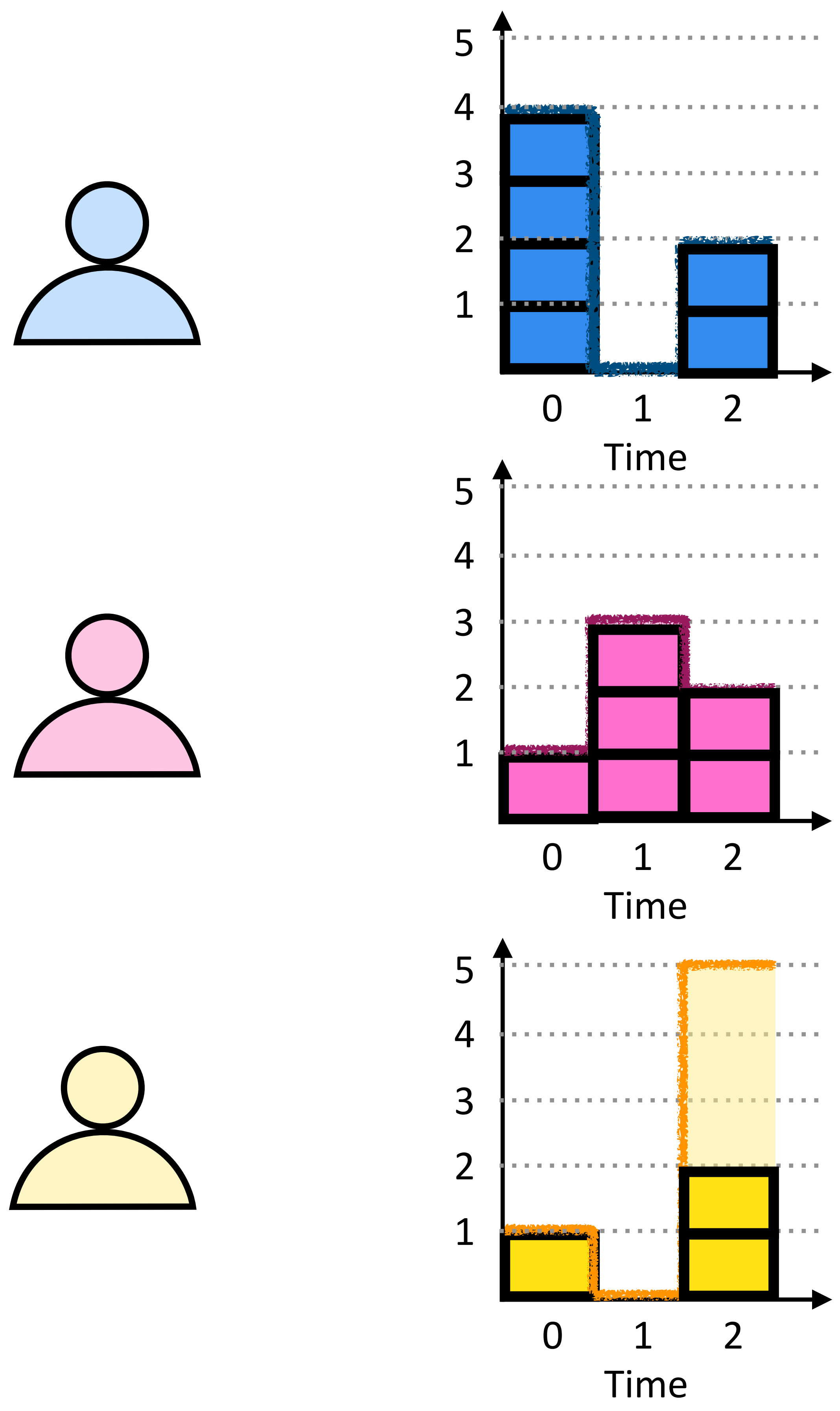
Fairness

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

Demands

Resource Pool

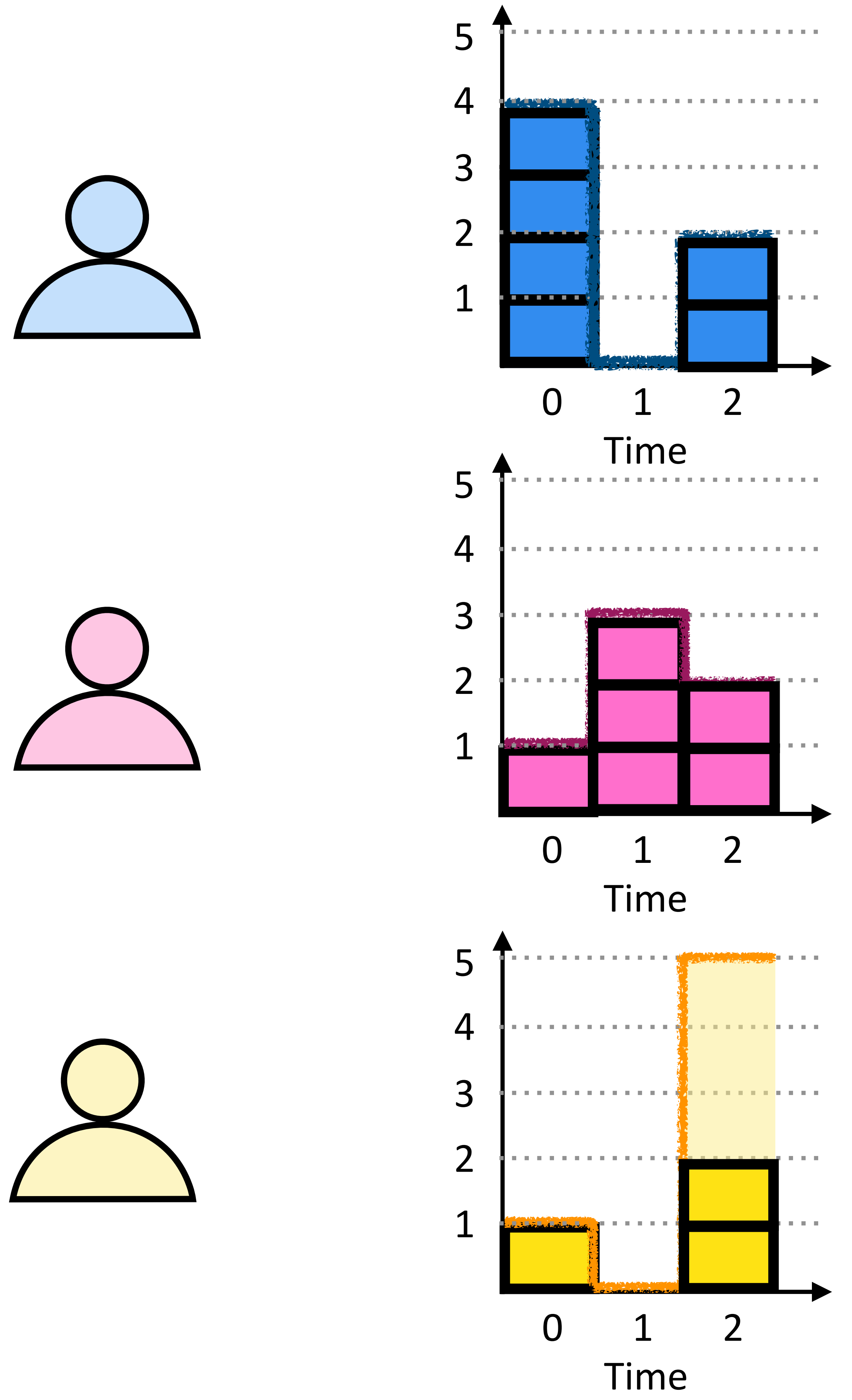


	Max-min fairness based on demands at t=0	Max-min fairness applied periodically
Pareto efficiency	✗	
Strategy-proofness	✗	
Fairness		

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

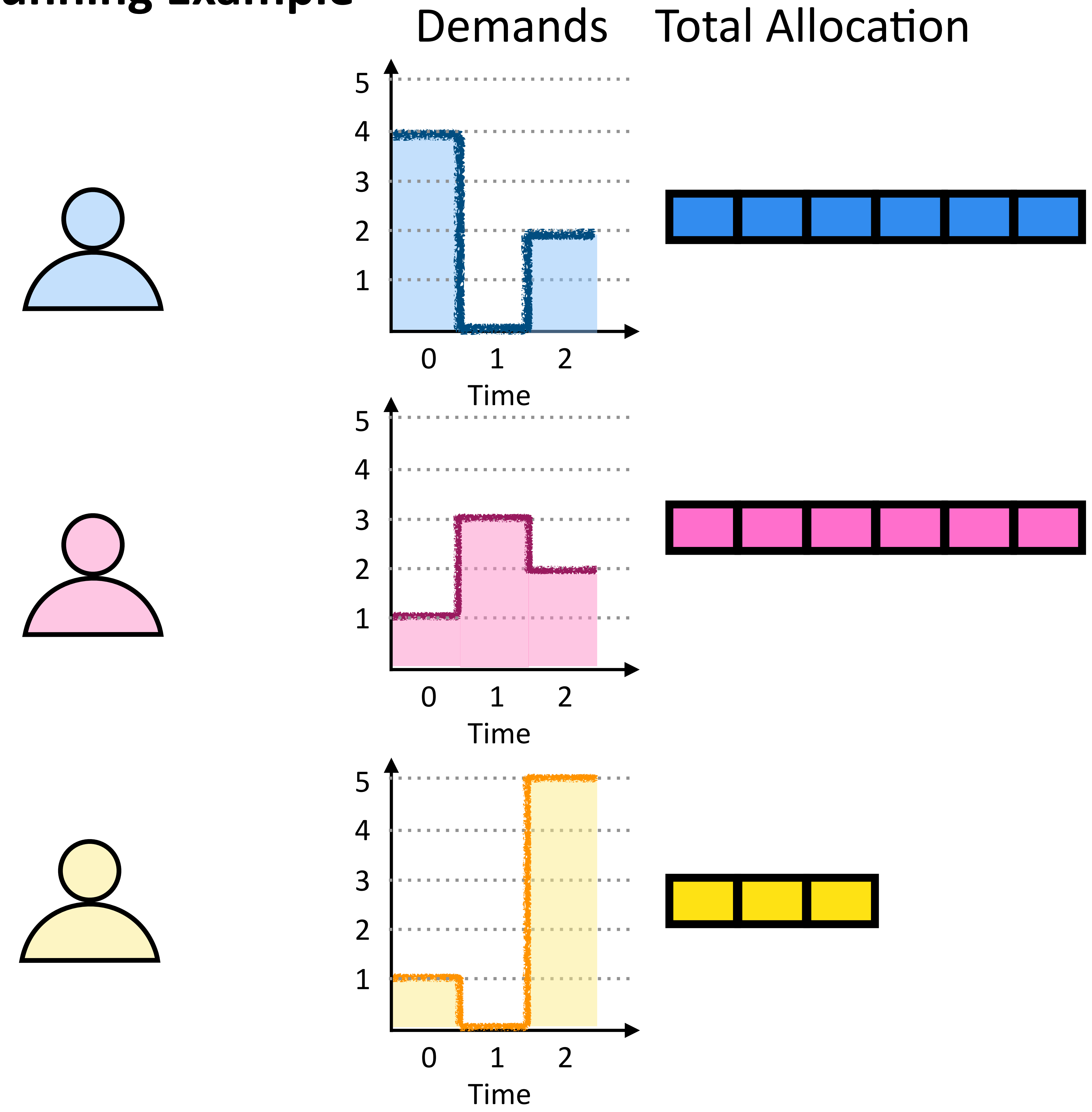
Demands   Total Allocation



	Max-min fairness based on demands at t=0	**Max-min fairness applied periodically**
Pareto efficiency	✗	
Strategy-proofness	✗	
Fairness		

Max-min fairness under dynamic demands: looses one or more of its properties

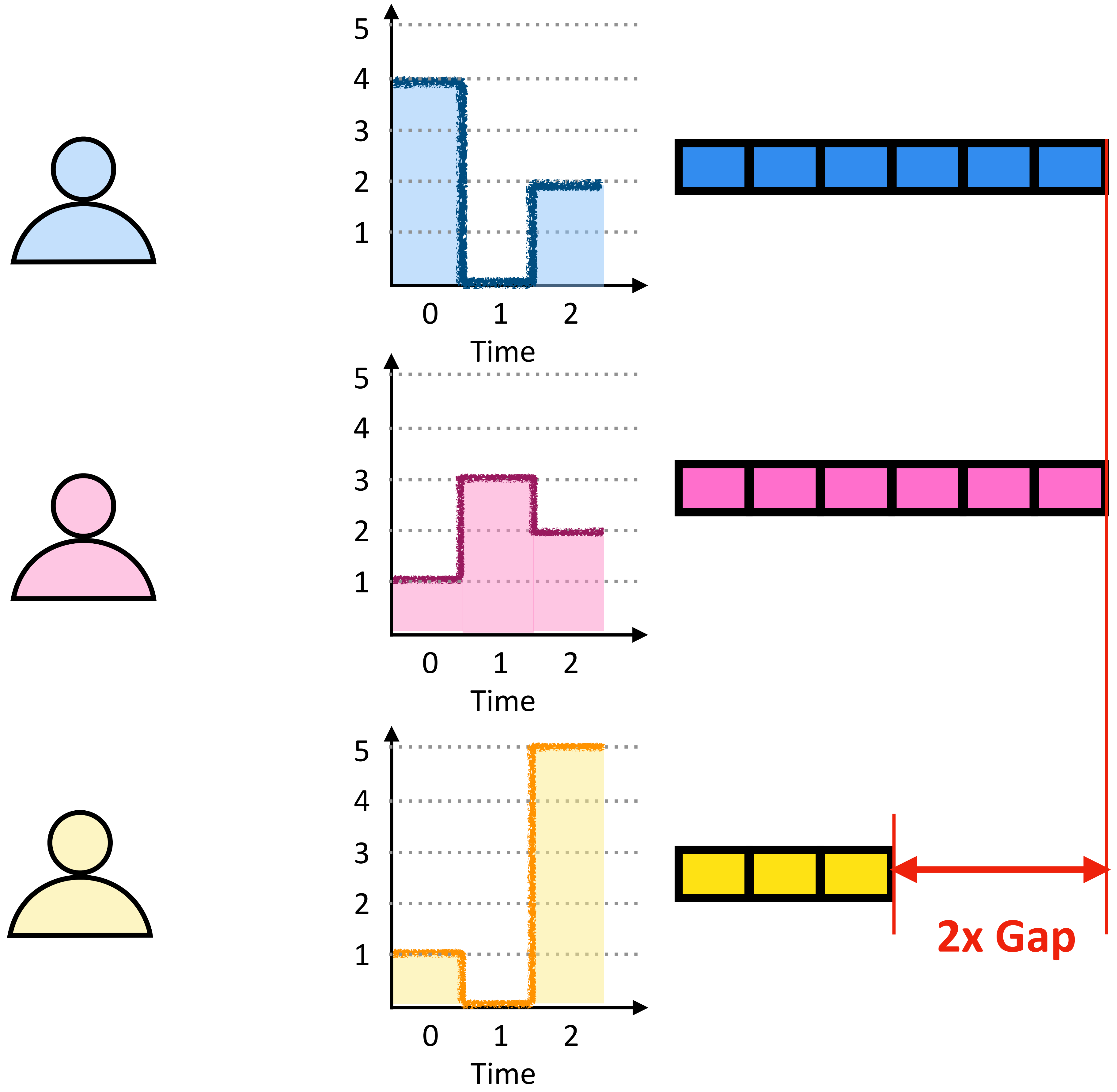
Running Example



	Max-min fairness based on demands at t=0	Max-min fairness applied periodically
Pareto efficiency	✗	
Strategy-proofness	✗	
Fairness		

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

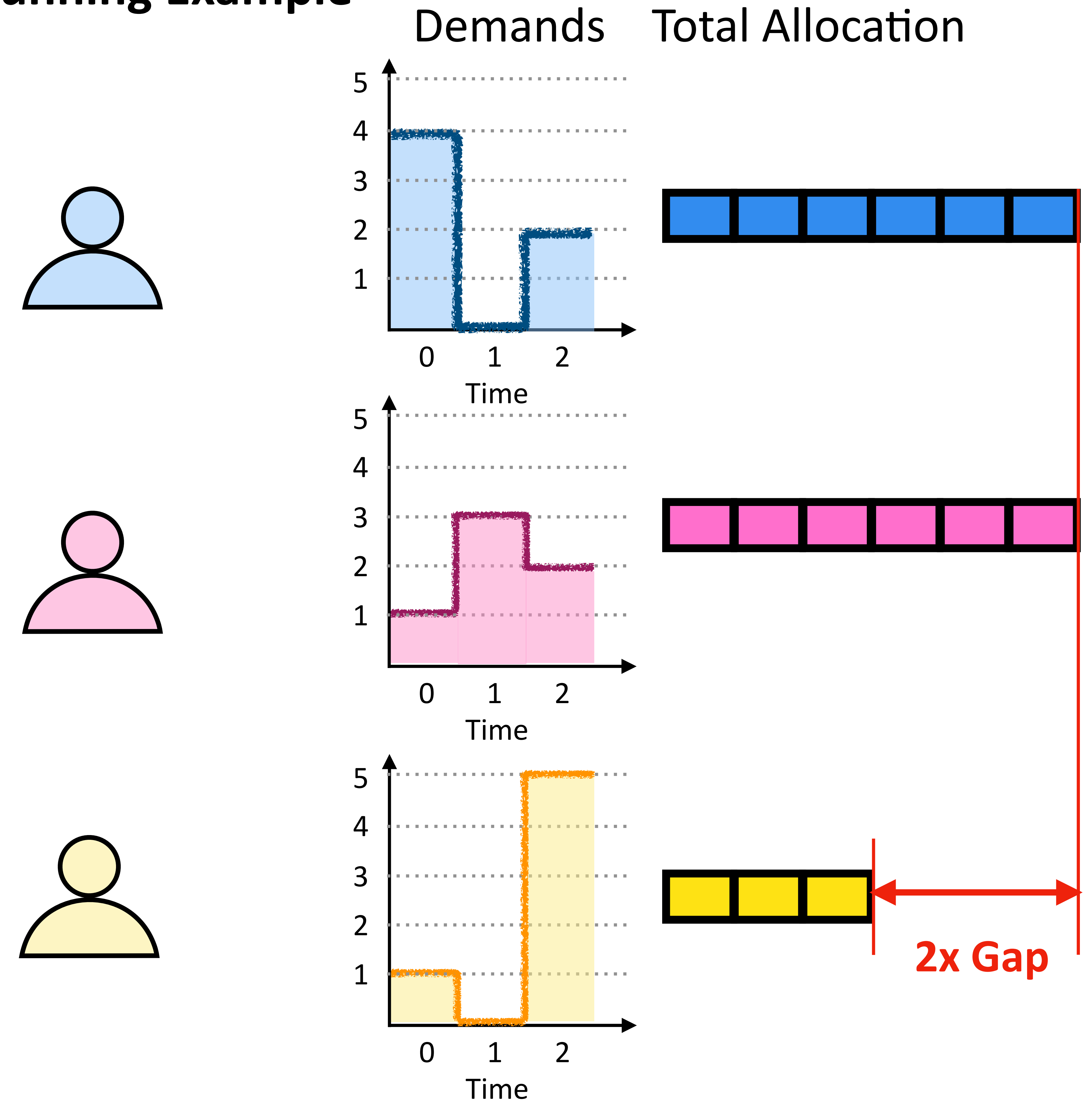


Max-min fairness based on demands at t=0

Max-min fairness applied periodically

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

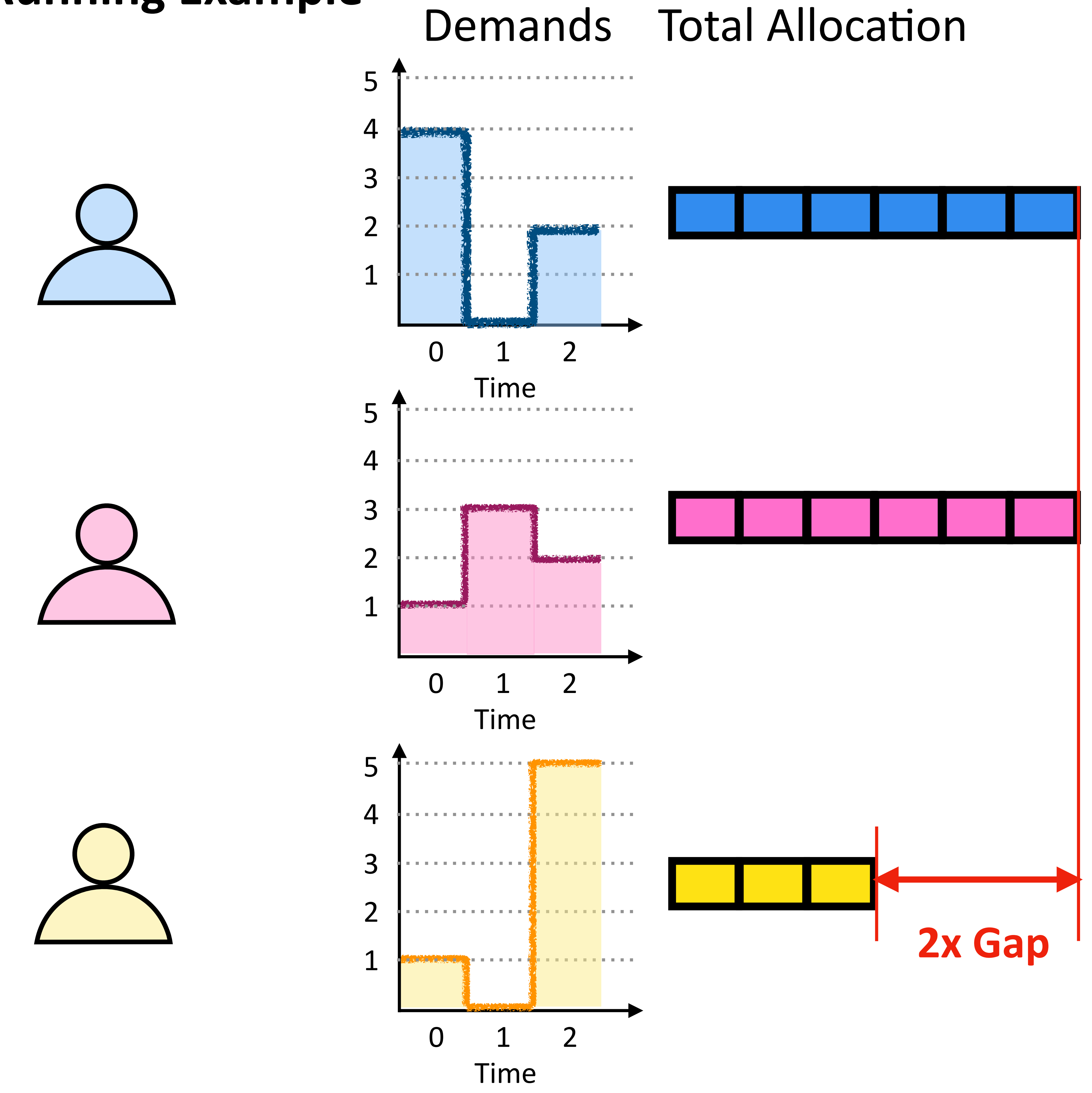


For n users, a user can get Ω(n) more allocation than others

	Max-min fairness based on demands at t=0	Max-min fairness applied periodically
Pareto efficiency	✗	
Strategy-proofness	✗	
Fairness		

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example



	Max-min fairness based on demands at t=0	Max-min fairness applied periodically
Pareto efficiency	✗	✓
Strategy-proofness	✗	✓
Fairness		✗

For n users, a user can get Ω(n) more allocation than others

Max-min fairness under dynamic demands: looses one or more of its properties

Running Example

For n users, a user can get Ω(n) more allocation than others

	Max-min fairness based on demands at t=0	Max-min fairness applied periodically
Pareto efficiency	✗	✓
Strategy-proofness	✗	✓
Fairness		✗

Max-min fairness under dynamic demands: looses one or more of its properties

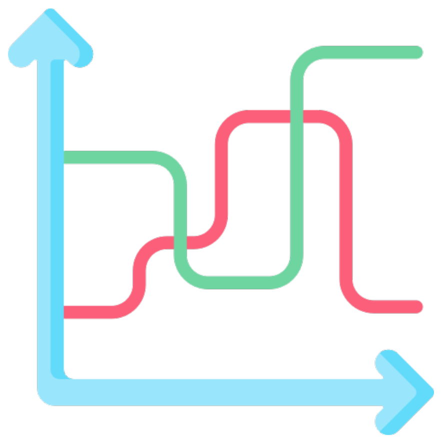
For n users, a user can get Ω(n) more allocation than others

	Max-min fairness based on demands at t=0	Max-min fairness applied periodically
Pareto efficiency	✗	✓
Strategy-proofness	✗	✓
Fairness		✗

Need to revisit classical resource allocation problem under dynamic demands

Karma: Revisiting the classical resource allocation problem under dynamic demands

Karma: Revisiting the classical resource allocation problem under dynamic demands



New resource allocation algorithm for dynamic demands

Karma: Revisiting the classical resource allocation problem under dynamic demands

New resource allocation algorithm for dynamic demands

Theoretical guarantees

- Pareto efficiency
- Strategy-proofness
- Fairness

Karma: Revisiting the classical resource allocation problem under dynamic demands

New resource allocation algorithm for dynamic demands

Theoretical guarantees

- Pareto efficiency
- Strategy-proofness
- Fairness

Prototype implementation and evaluation

Karma: Revisiting the classical resource allocation problem under dynamic demands

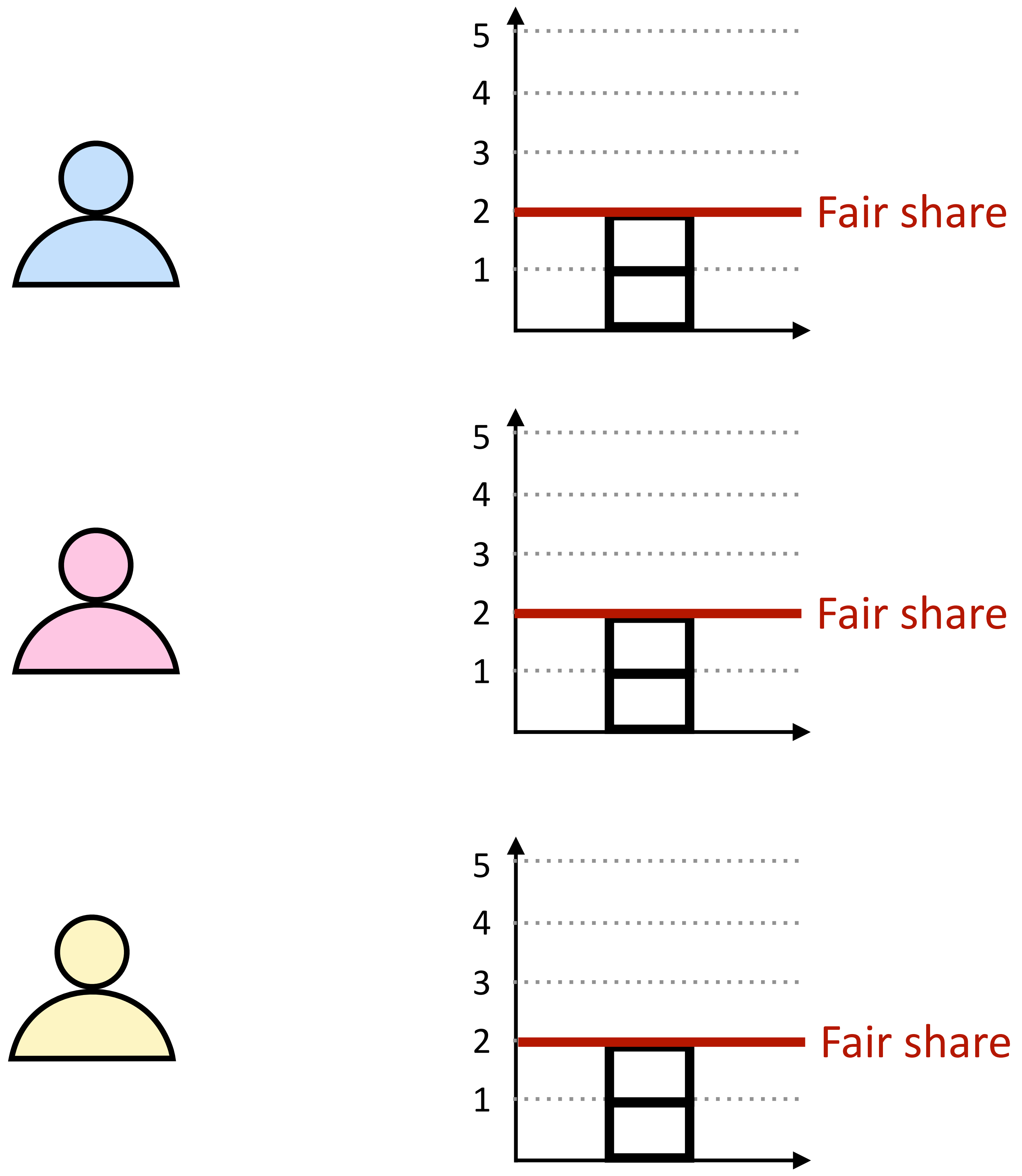
New resource allocation algorithm for dynamic demands

Theoretical guarantees

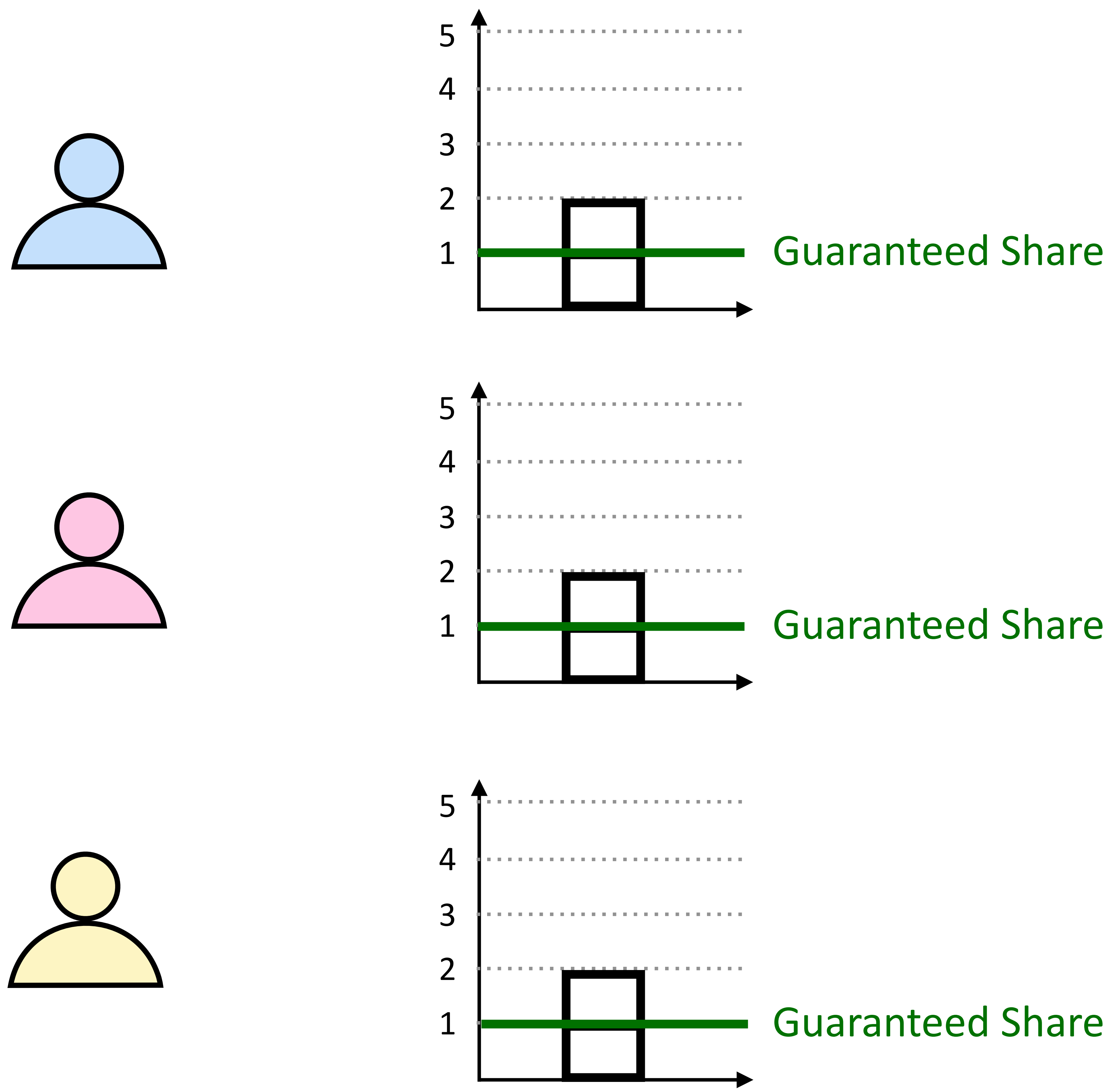
- Pareto efficiency
- Strategy-proofness
- Fairness

Prototype implementation and evaluation

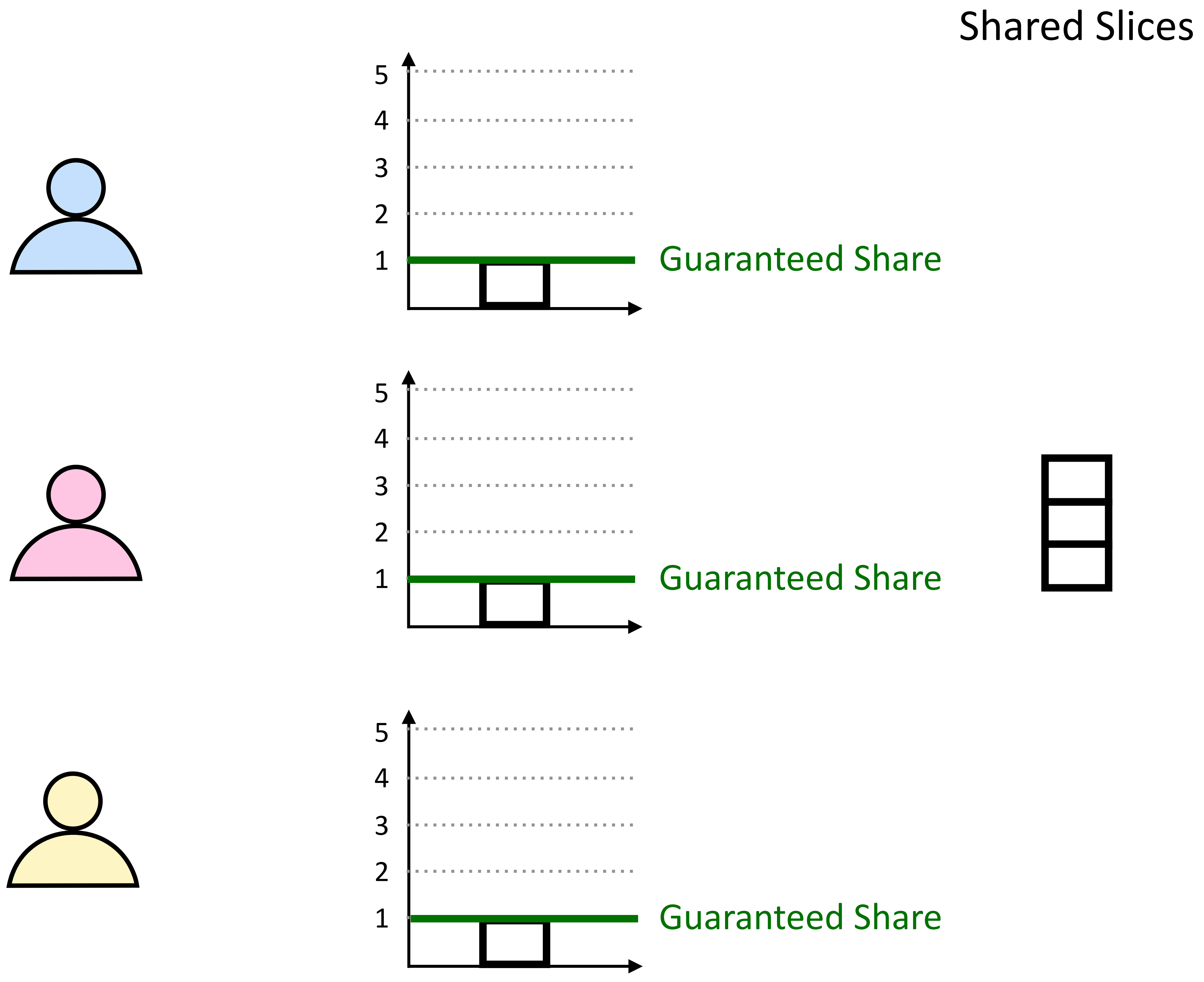
Karma key idea #1: *donated slices* and *shared slices*



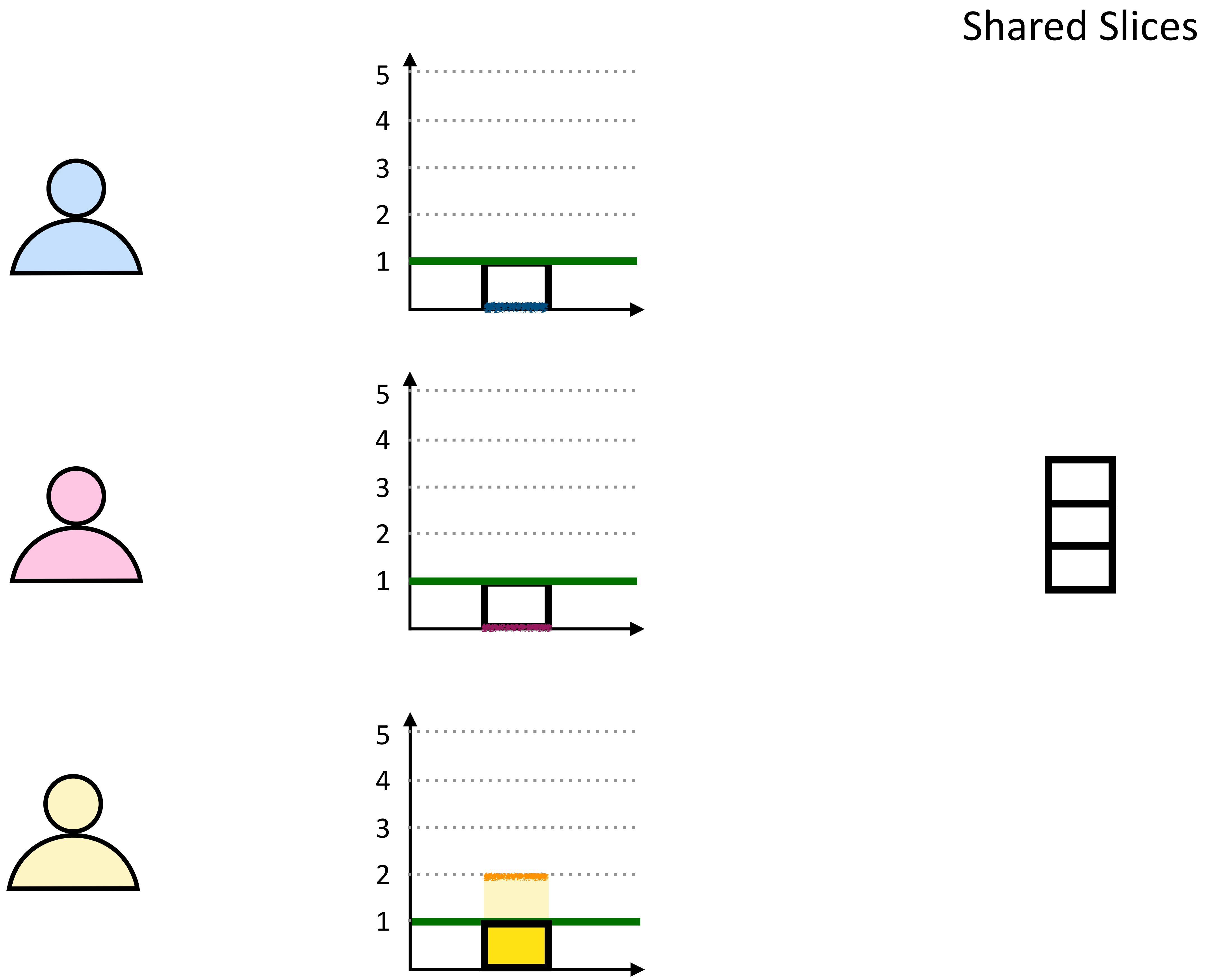
Karma key idea #1: *donated slices* and *shared slices*



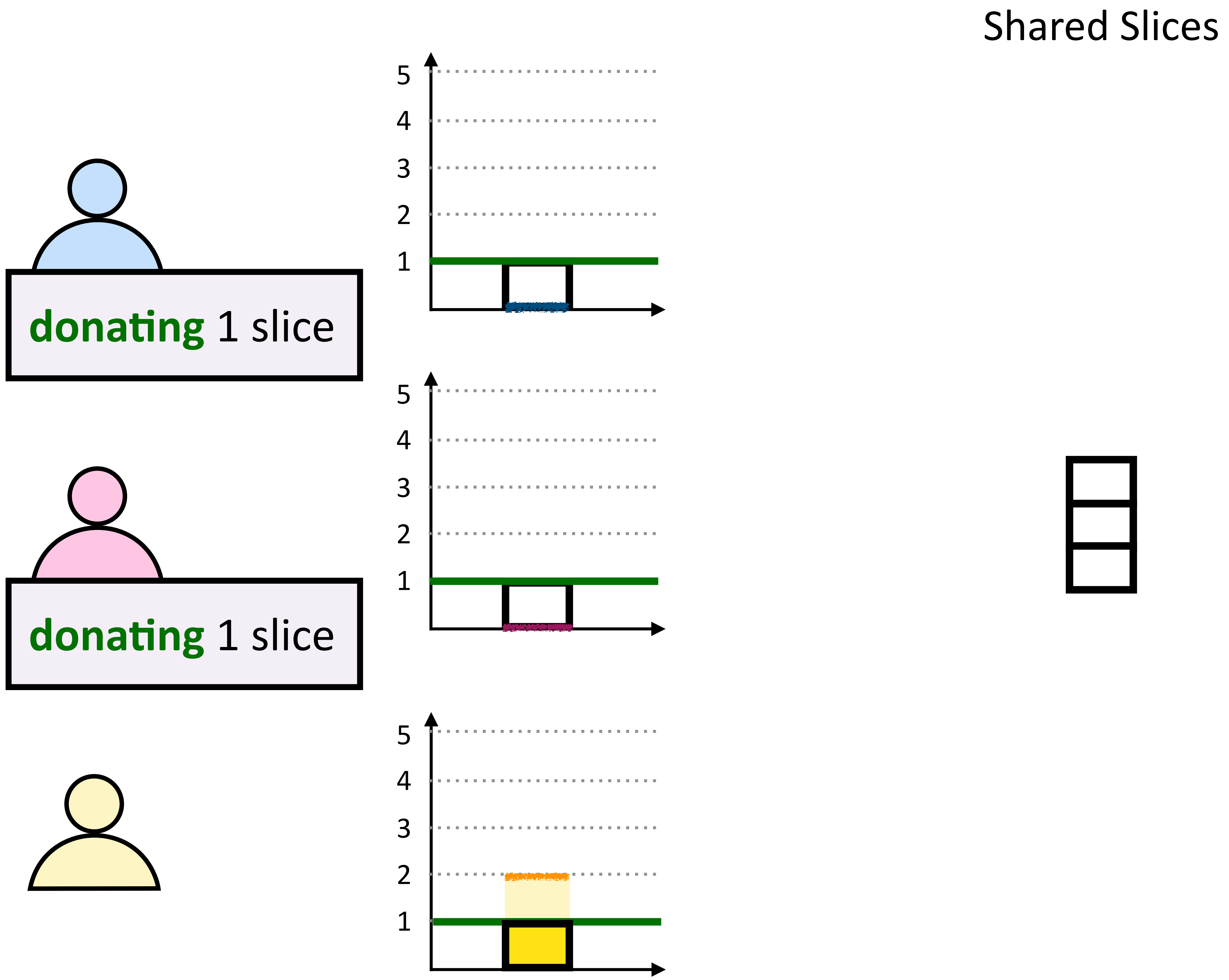
Karma key idea #1: *donated slices* and *shared slices*



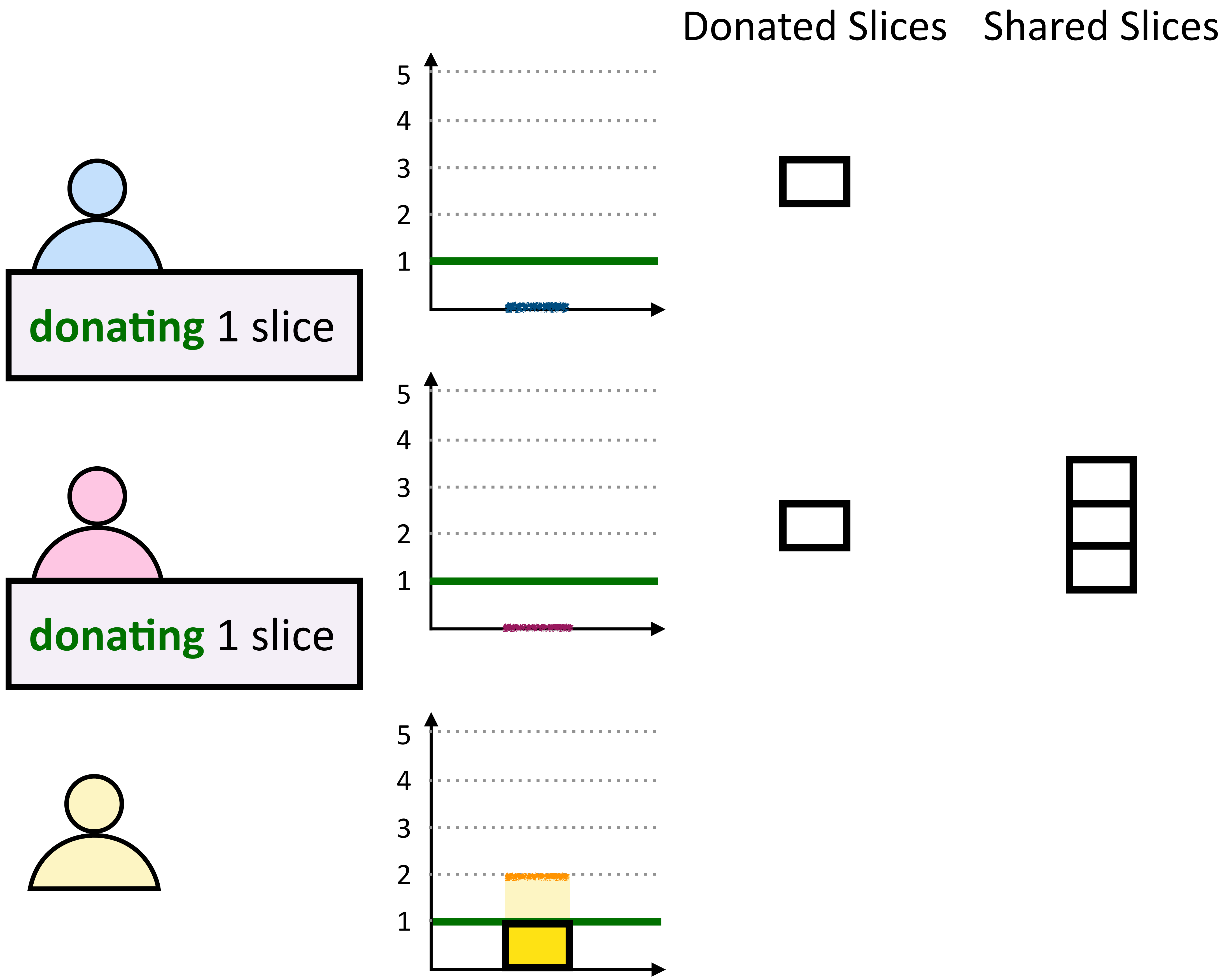
Karma key idea #1: *donated slices* and *shared slices*



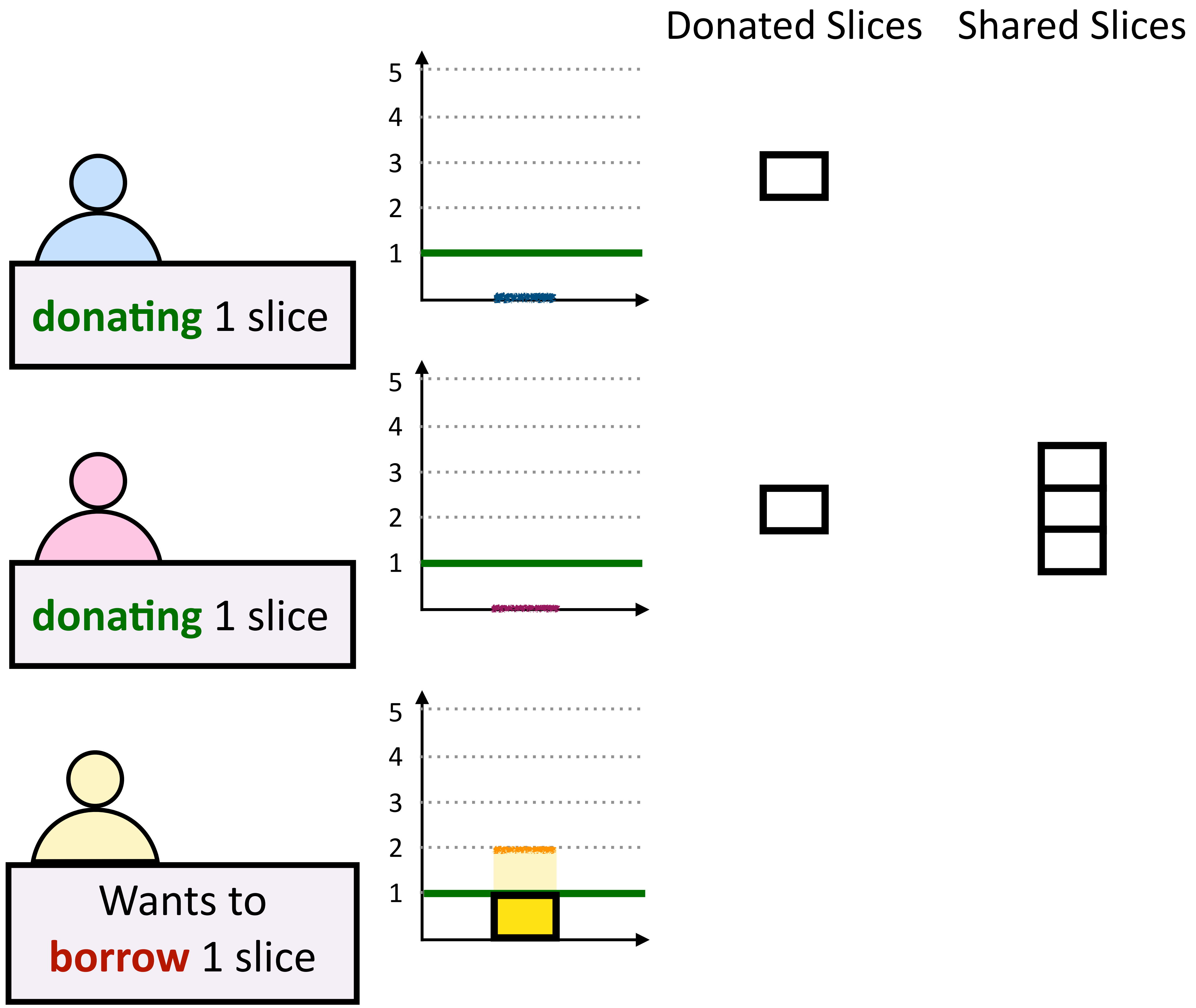
Karma key idea #1: *donated slices* and *shared slices*



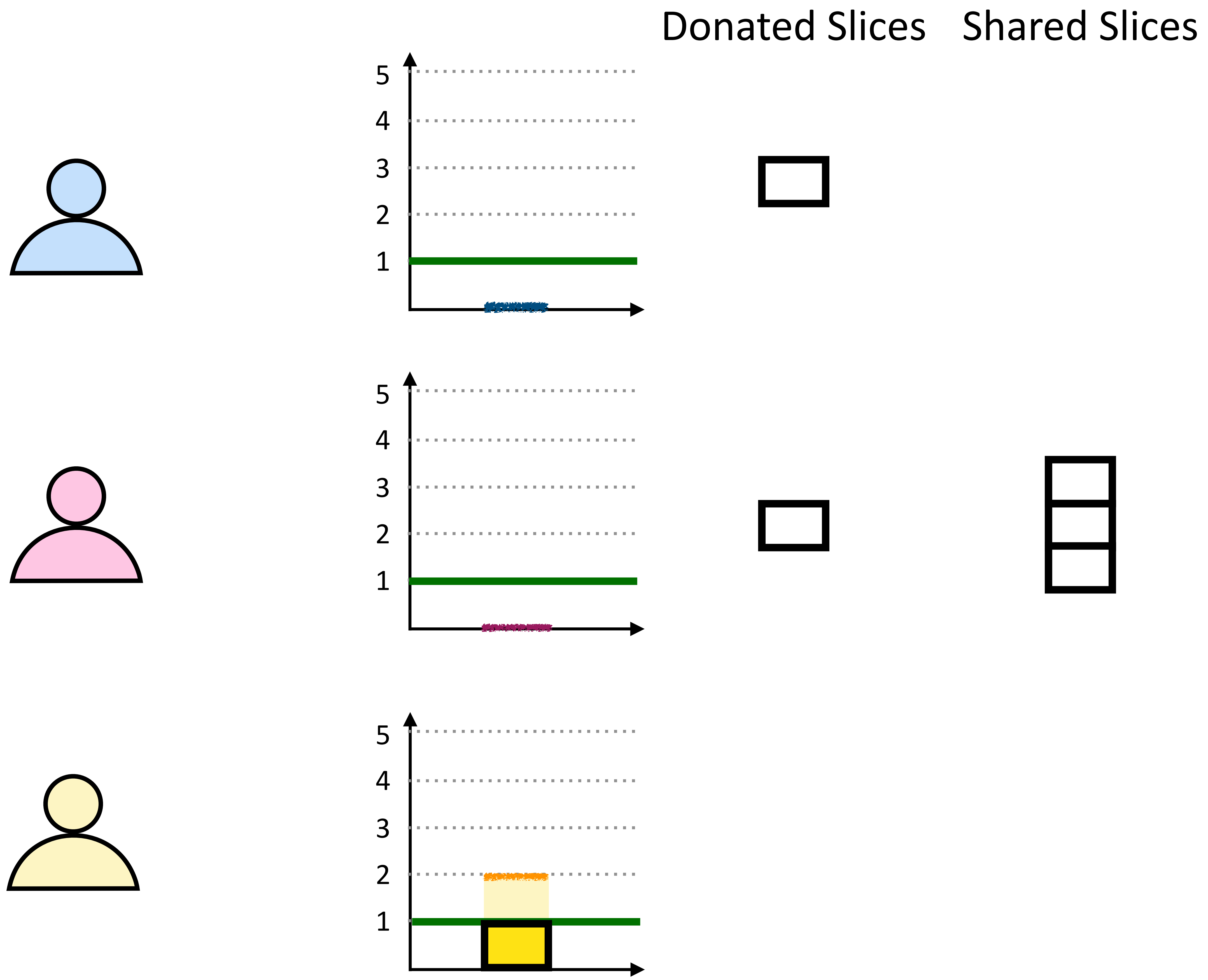
Karma key idea #1: *donated slices* and *shared slices*



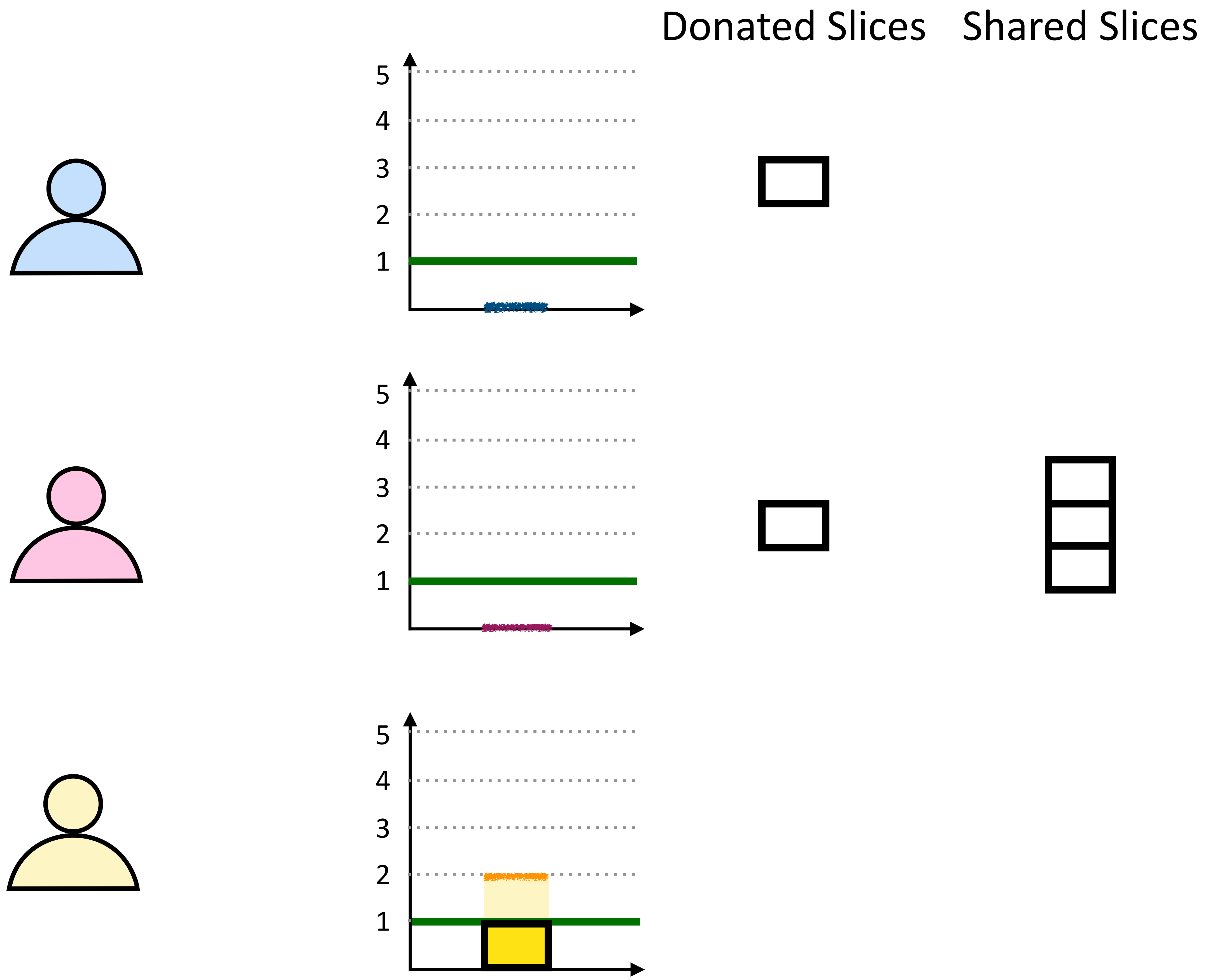
Karma key idea #1: *donated slices* and *shared slices*



Karma key idea #2: "credits" for resource allocation

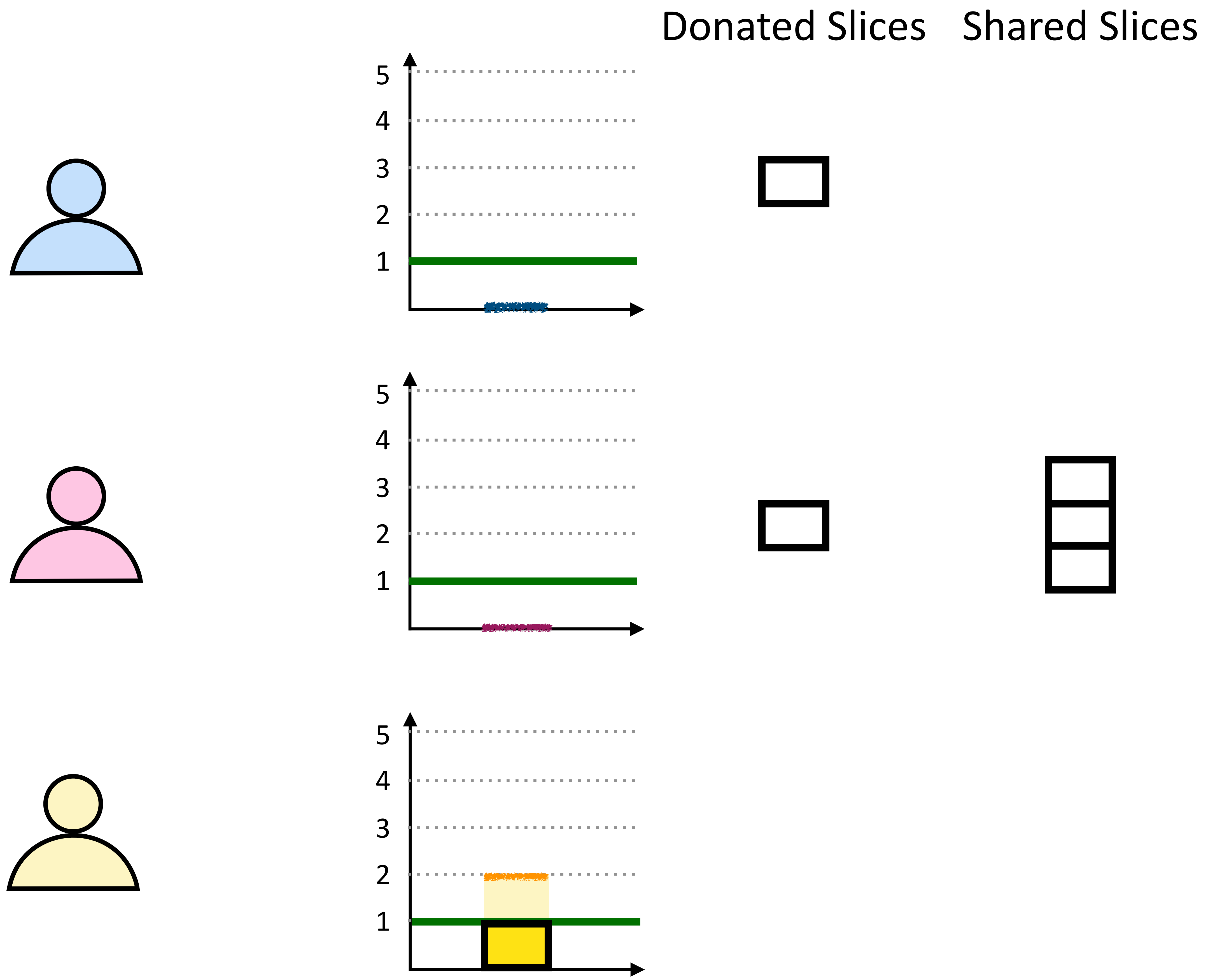


Karma key idea #2: “credits” for resource allocation



-1 credit for **borrowing** a slice

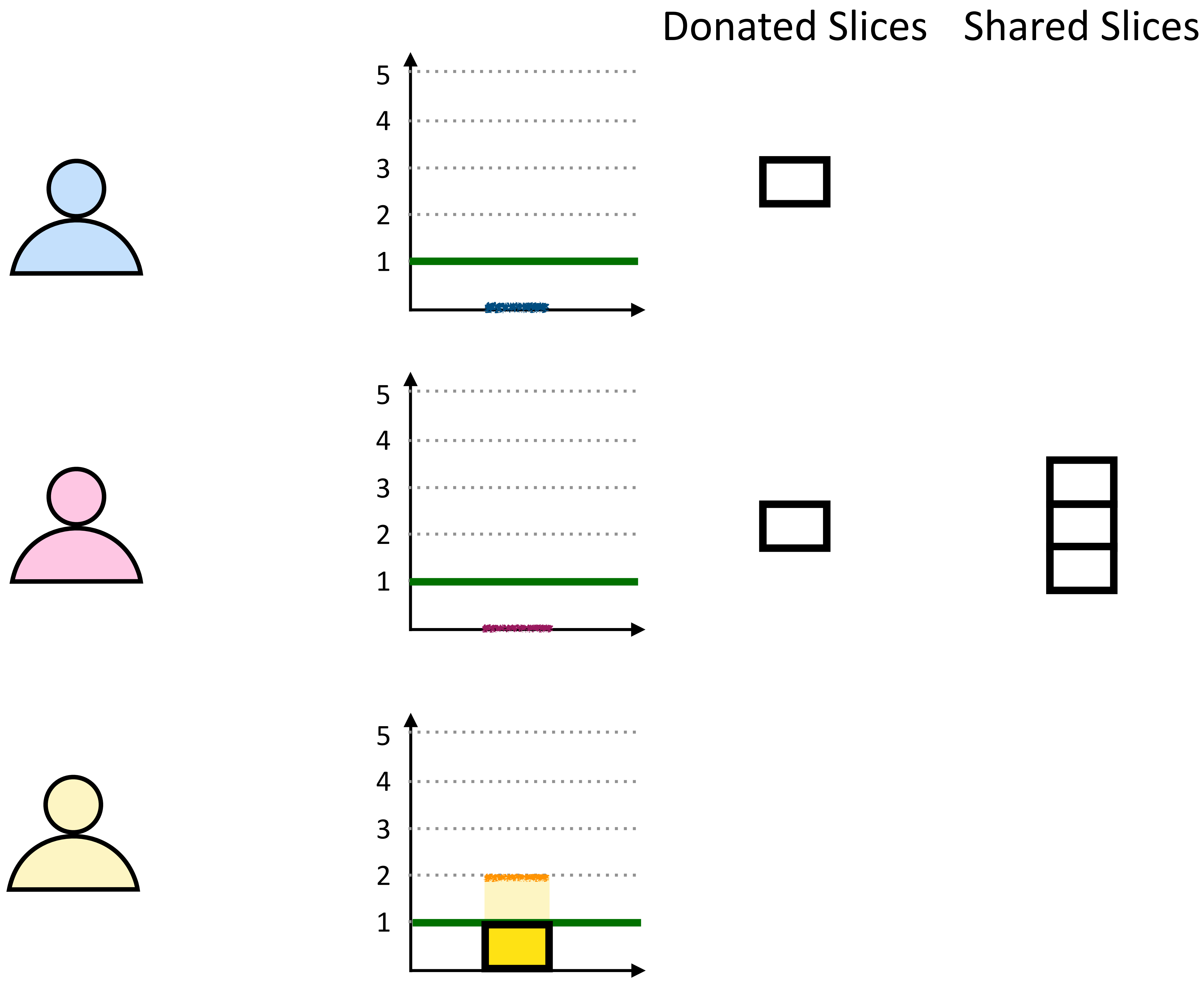
Karma key idea #2: “credits” for resource allocation



-1 credit for **borrowing** a slice

+1 credit if a **donated** slice is borrowed

Karma key idea #2: “credits” for resource allocation

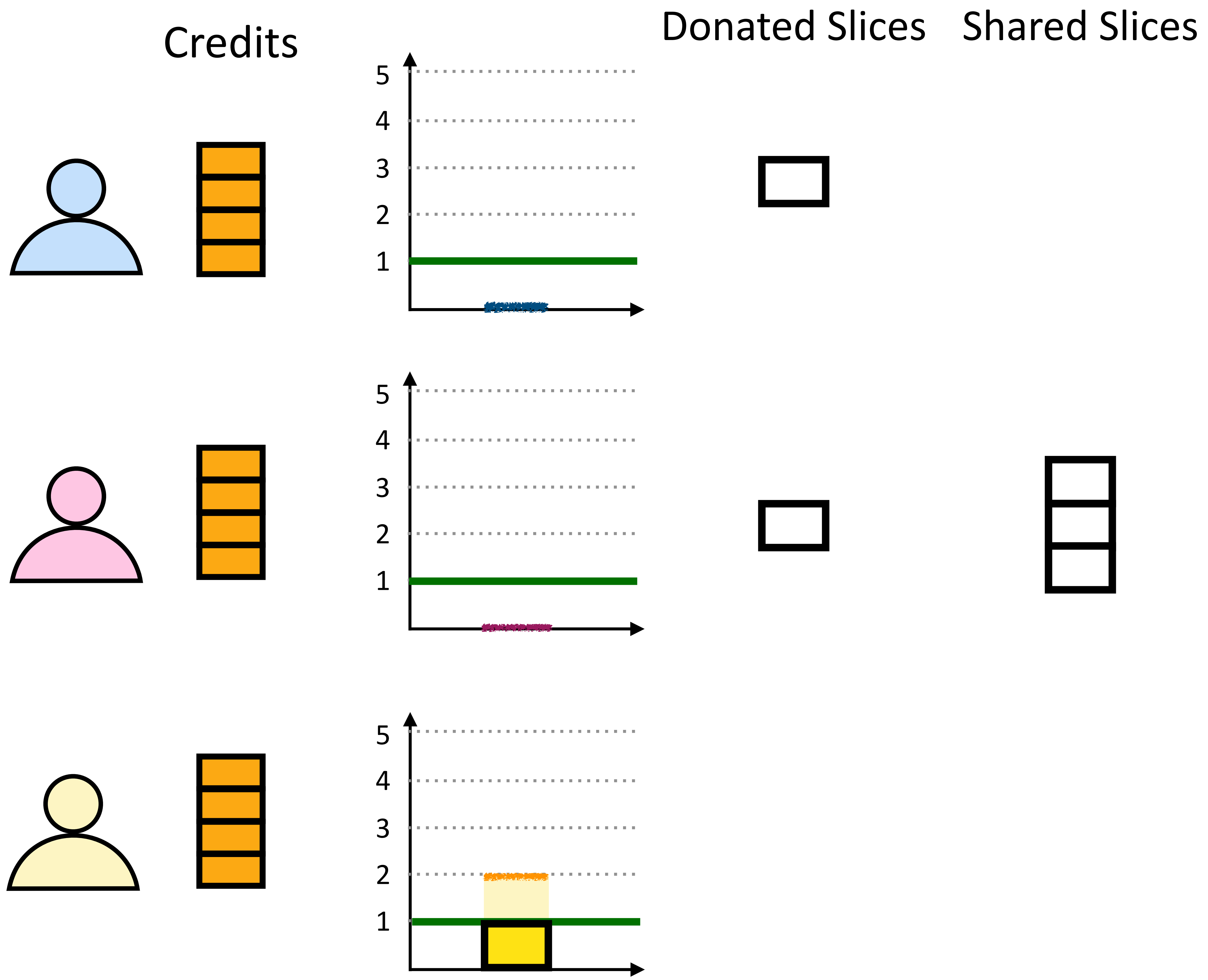


-1 credit for **borrowing** a slice

+1 credit if a **donated** slice is borrowed

(Every user is given initial credits at t=0)

Karma key idea #2: “credits” for resource allocation

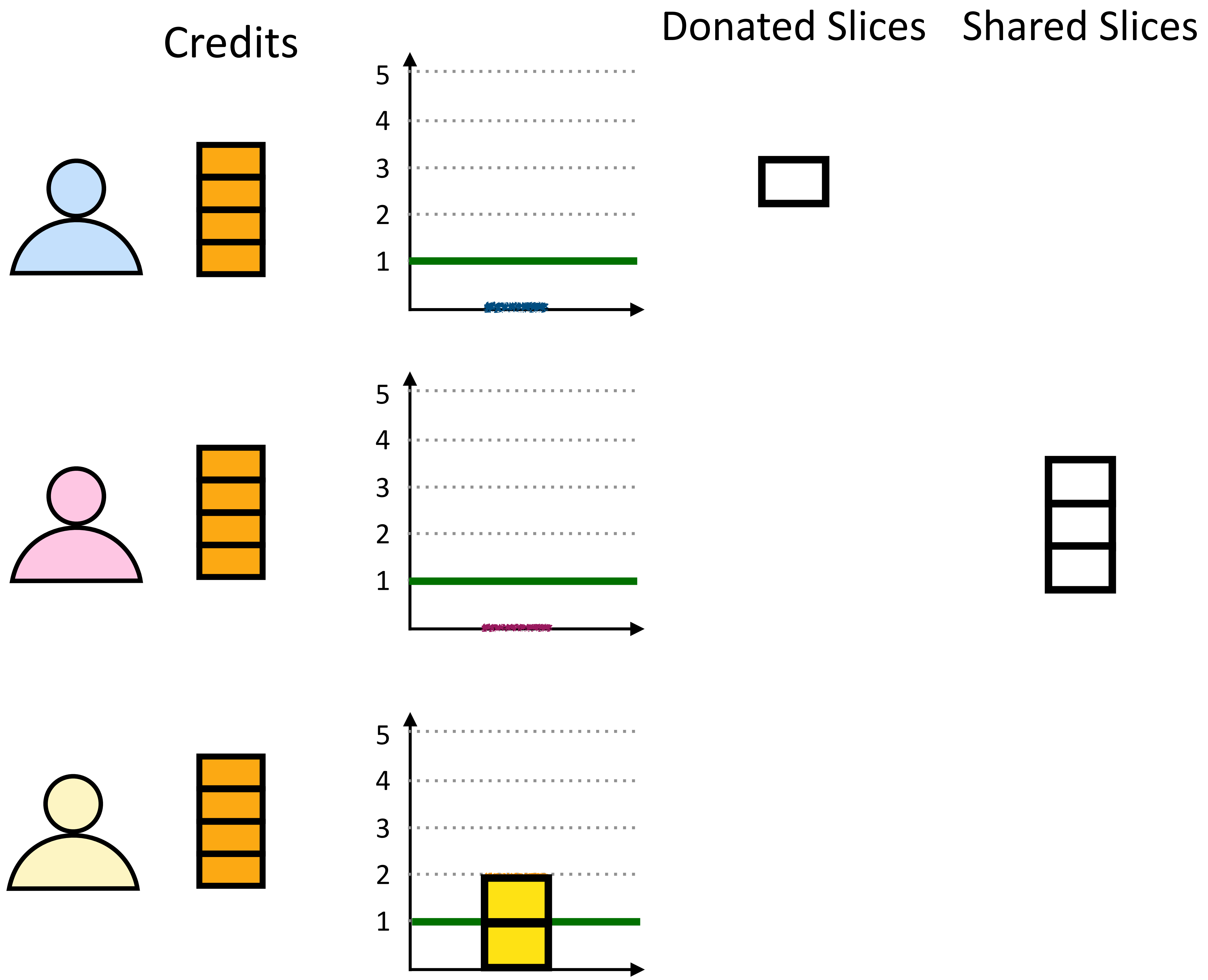


-1 credit for **borrowing** a slice

+1 credit if a **donated** slice is borrowed

(Every user is given initial credits at t=0)

Karma key idea #2: “credits” for resource allocation

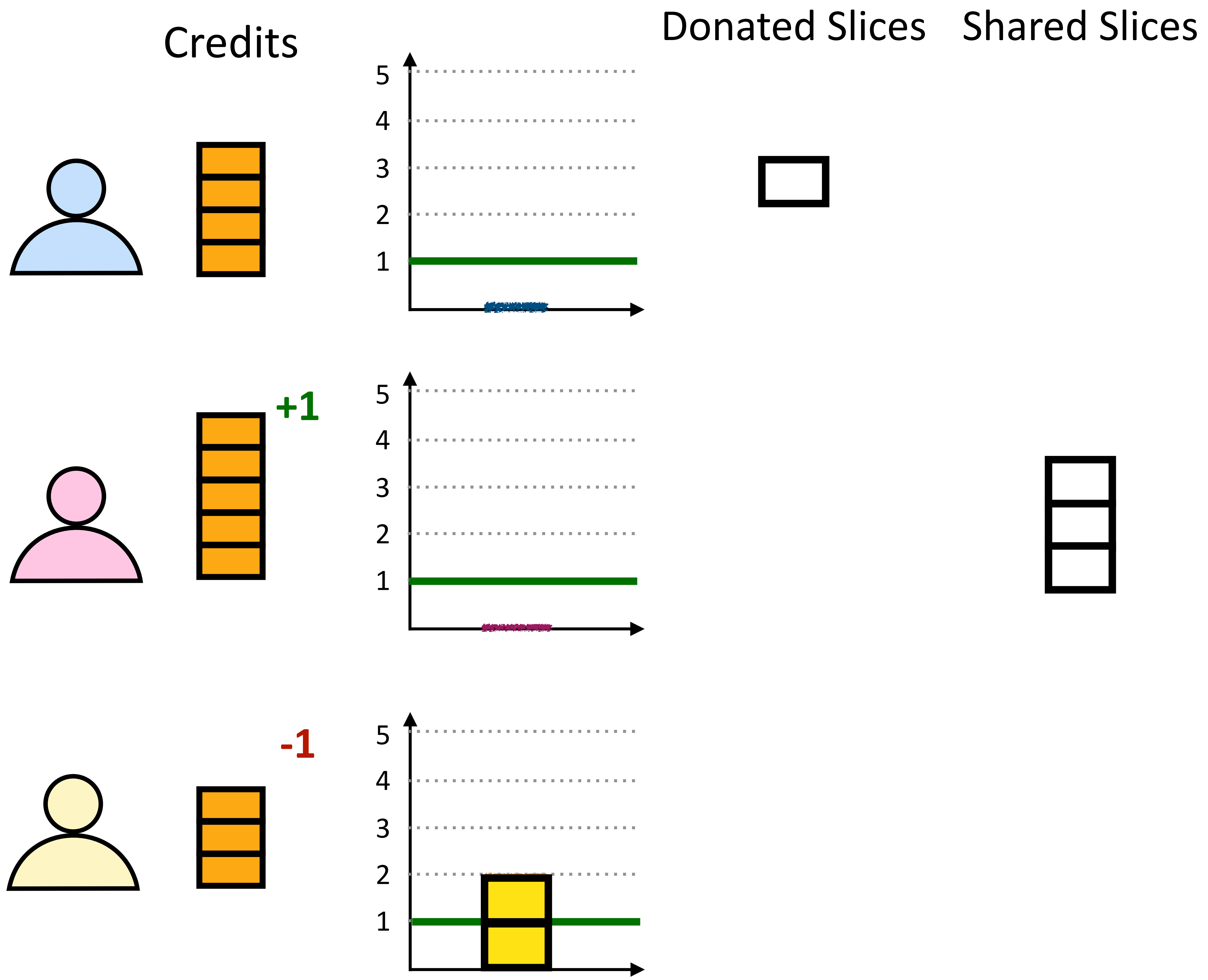


-1 credit for **borrowing** a slice

+1 credit if a **donated** slice is borrowed

(Every user is given initial credits at t=0)

Karma key idea #2: “credits” for resource allocation



-1 credit for **borrowing** a slice

+1 credit if a **donated** slice is borrowed

(Every user is given initial credits at t=0)

Karma allocation algorithm

Karma allocation algorithm

Pick **borrower** with **maximum** credits

Karma allocation algorithm

Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

Karma allocation algorithm

Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

Karma allocation algorithm

Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

Allocate slice to borrower

Karma allocation algorithm

Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

Karma allocation algorithm

Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

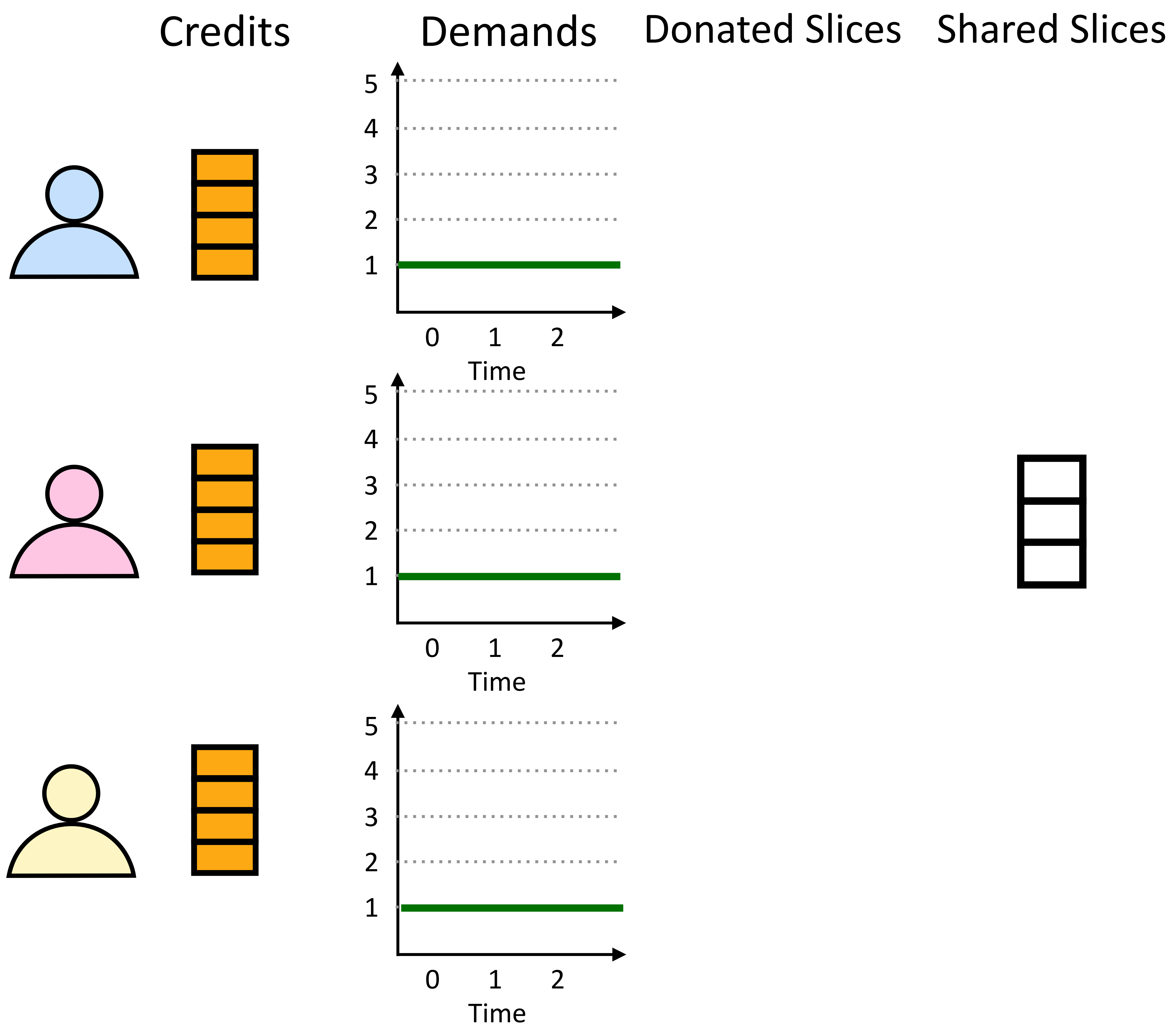
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

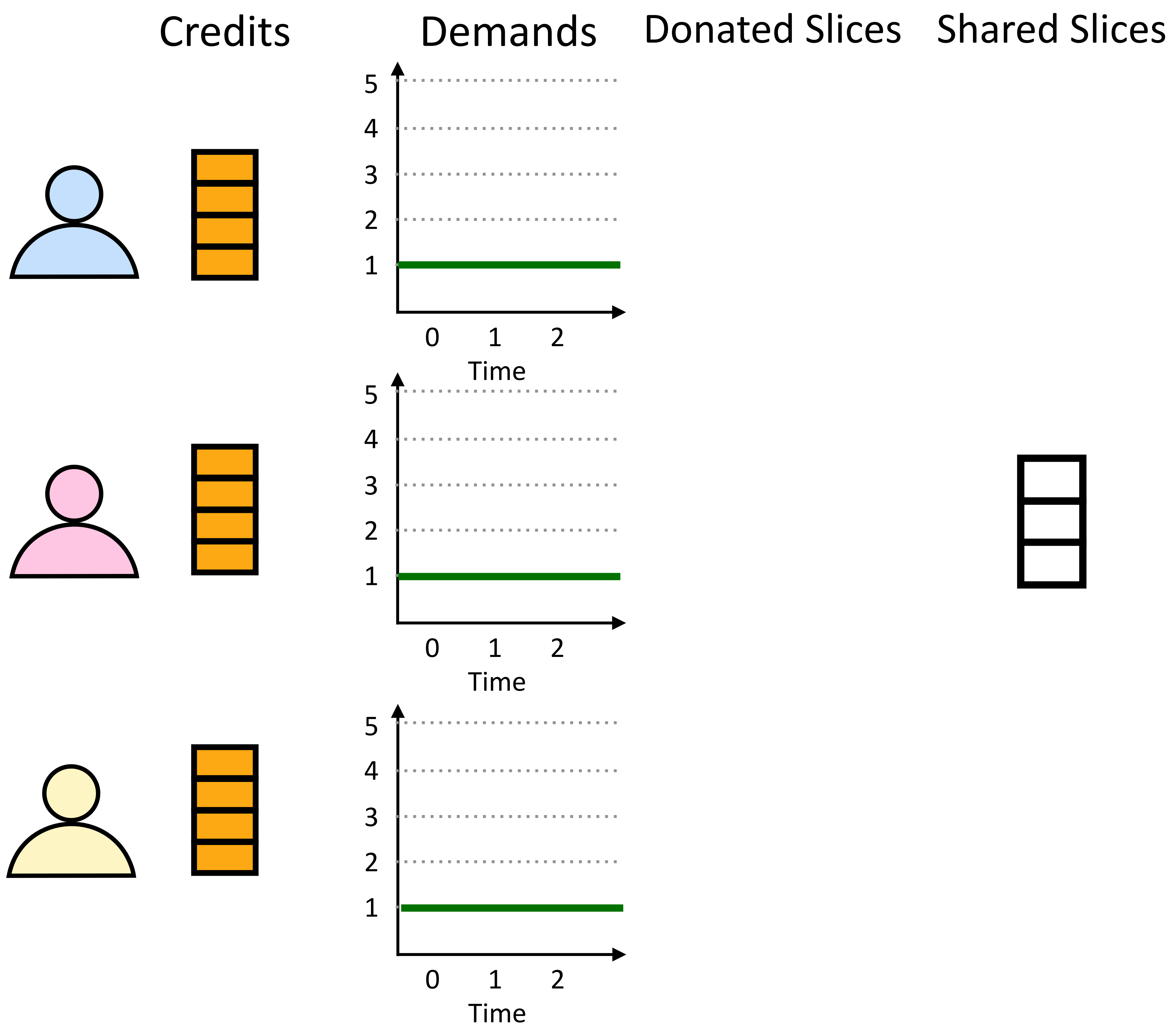
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

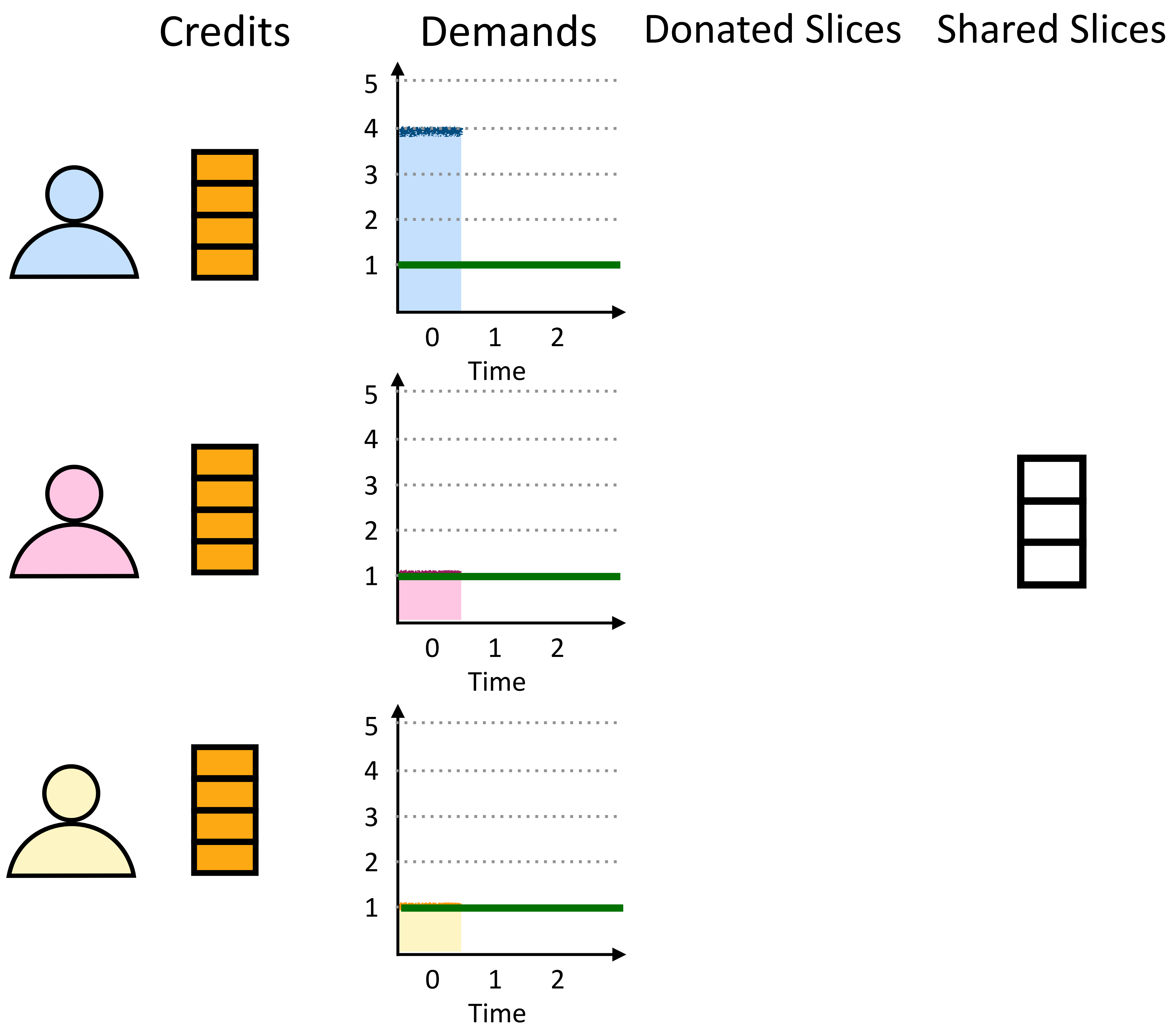
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits
If no donors, then use a shared slice

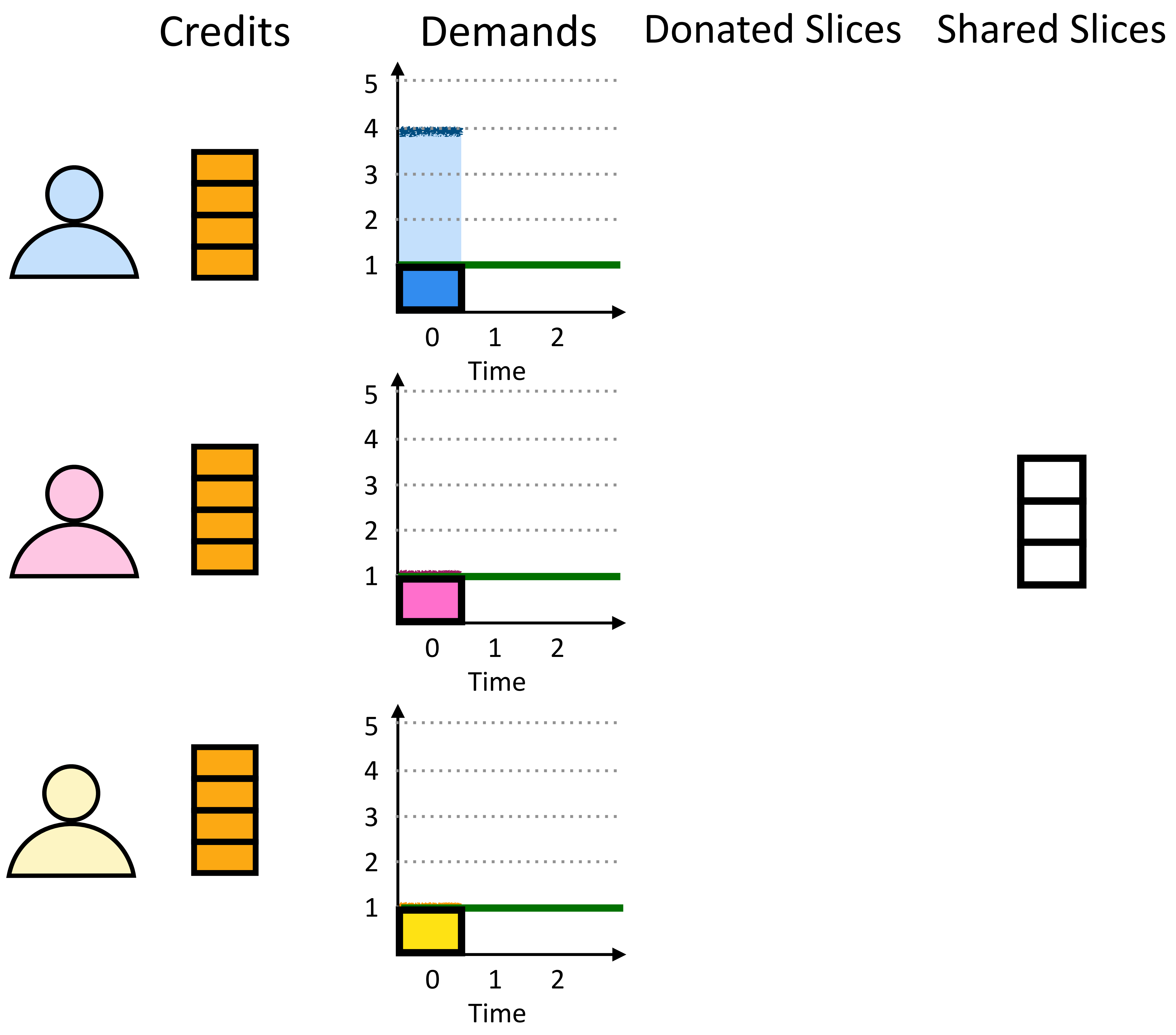
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

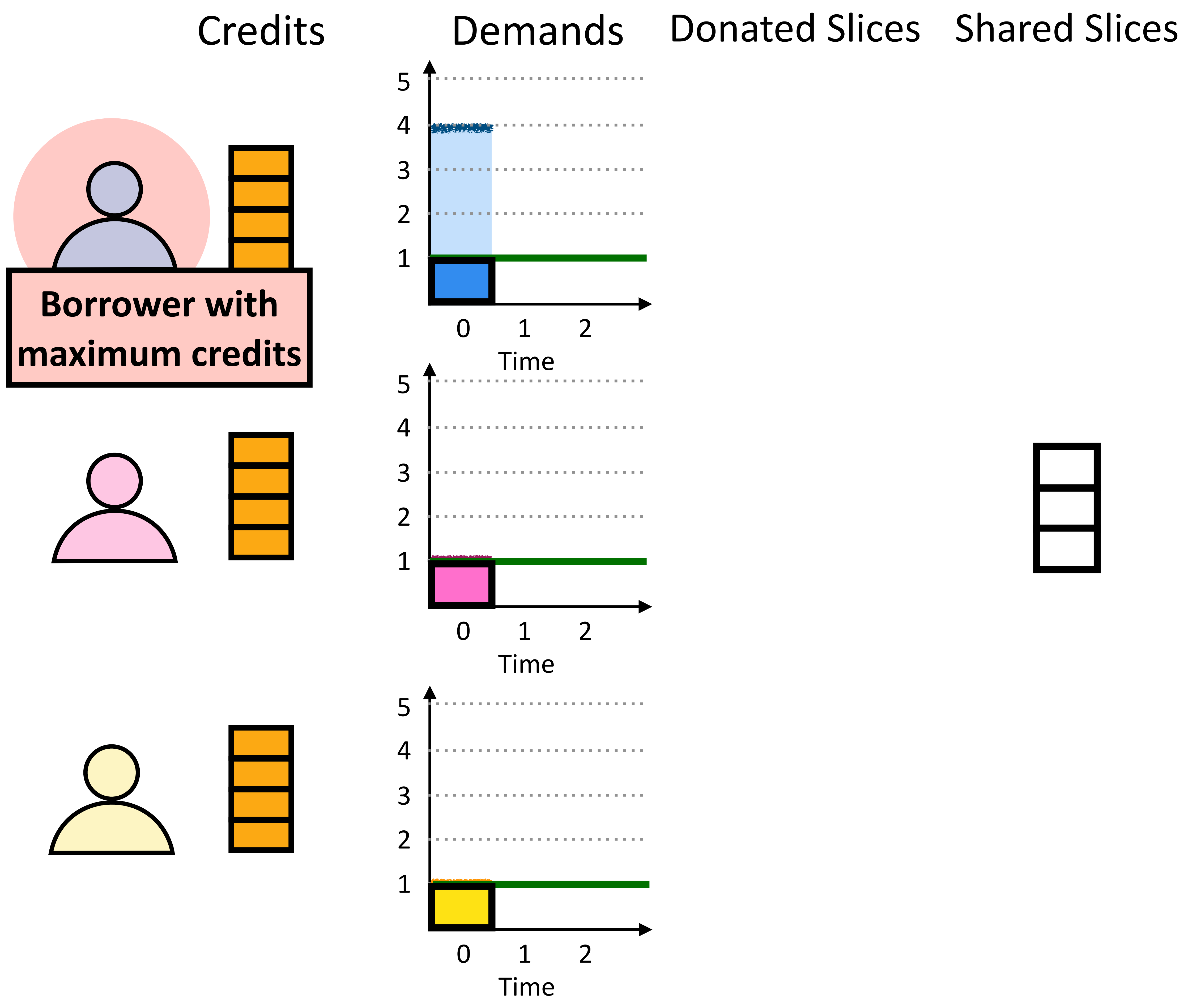
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

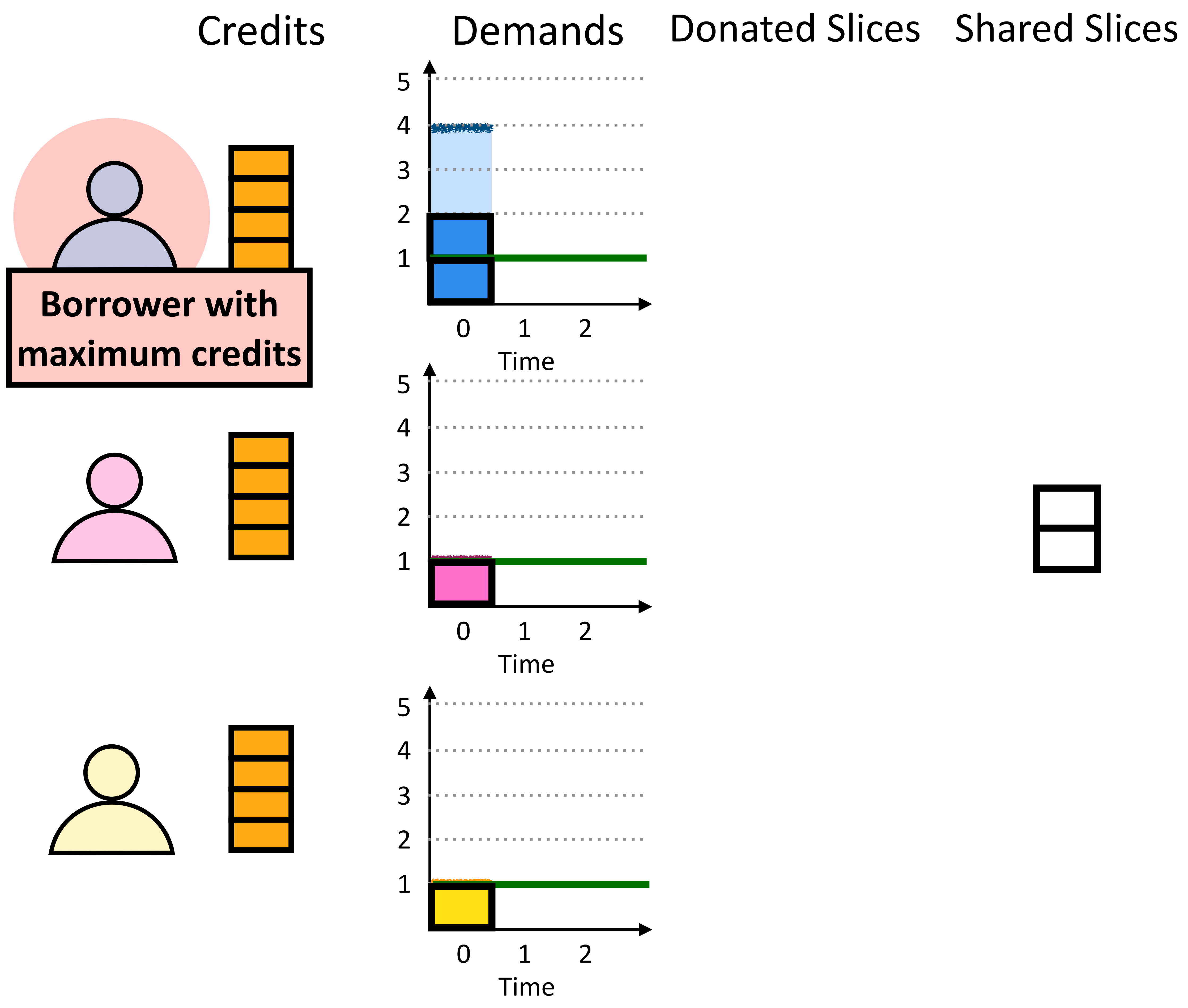
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

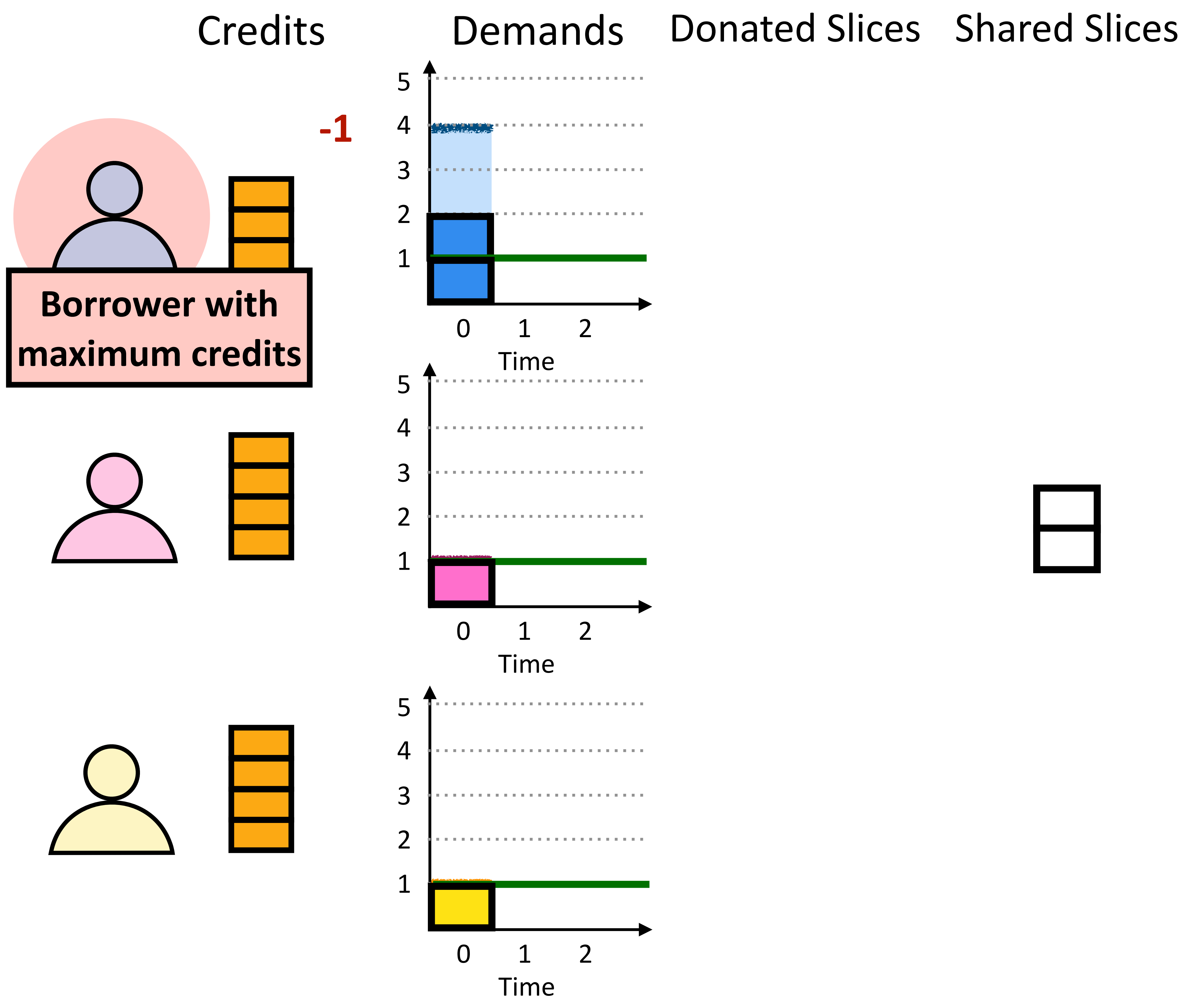
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

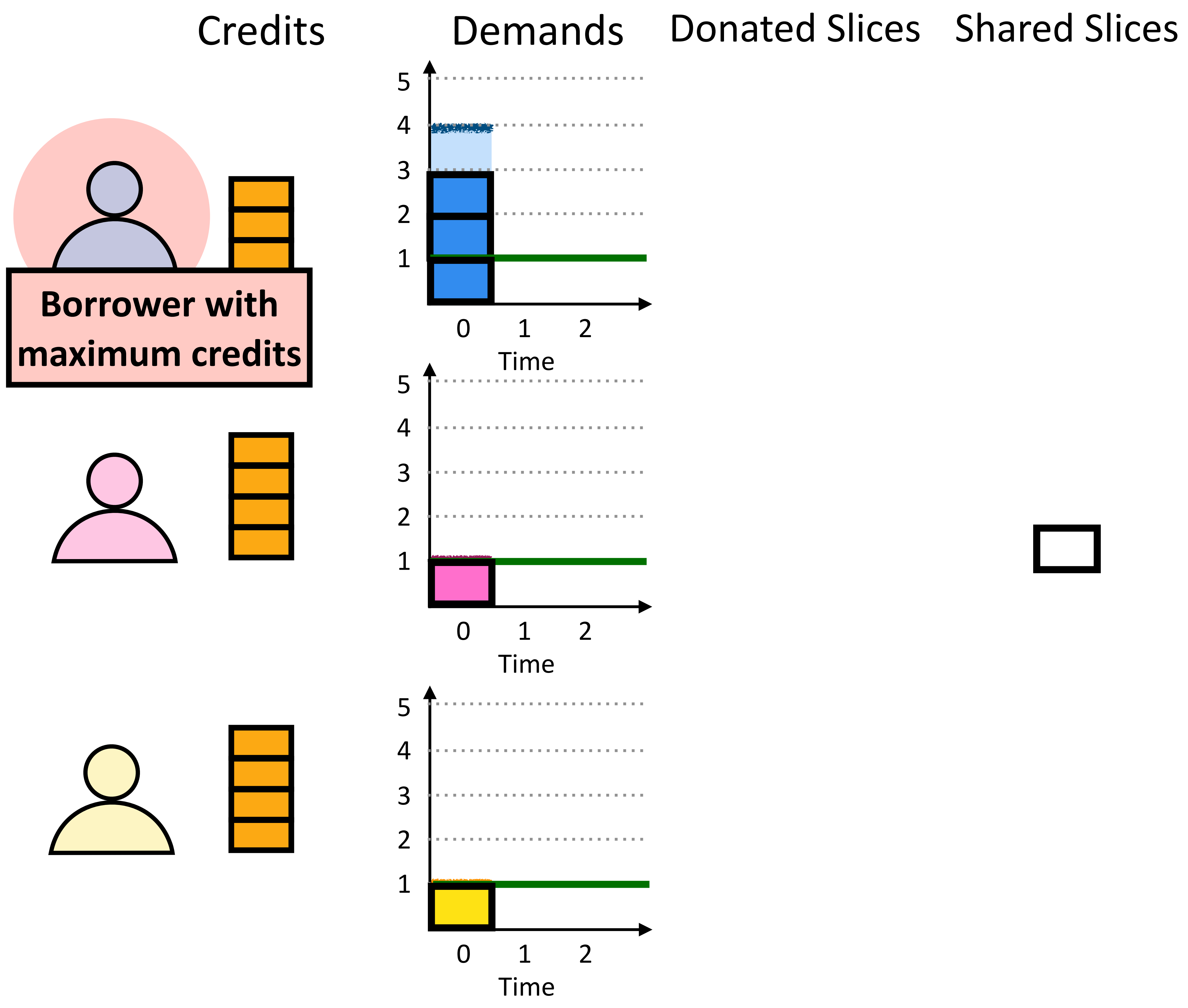
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

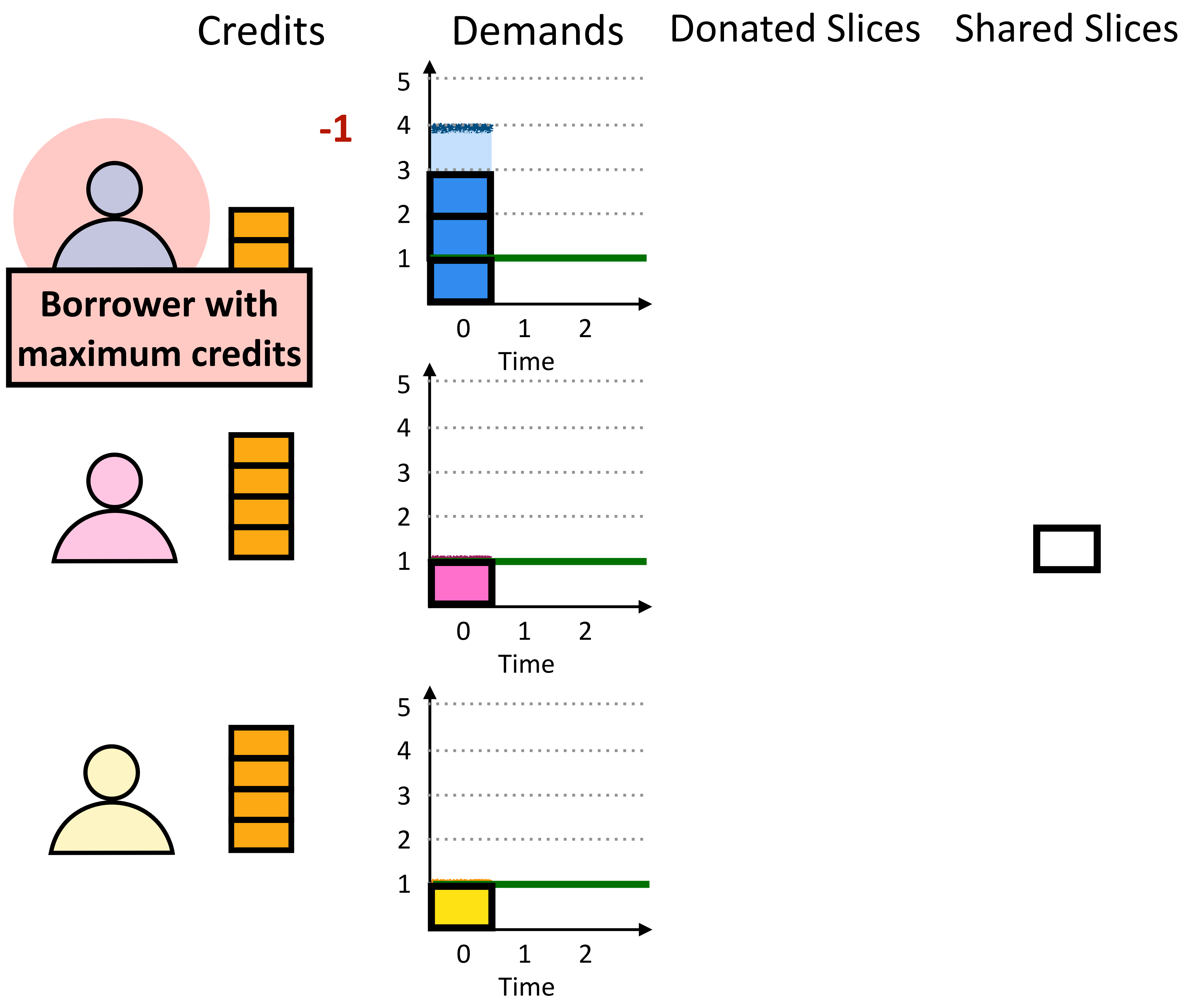
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

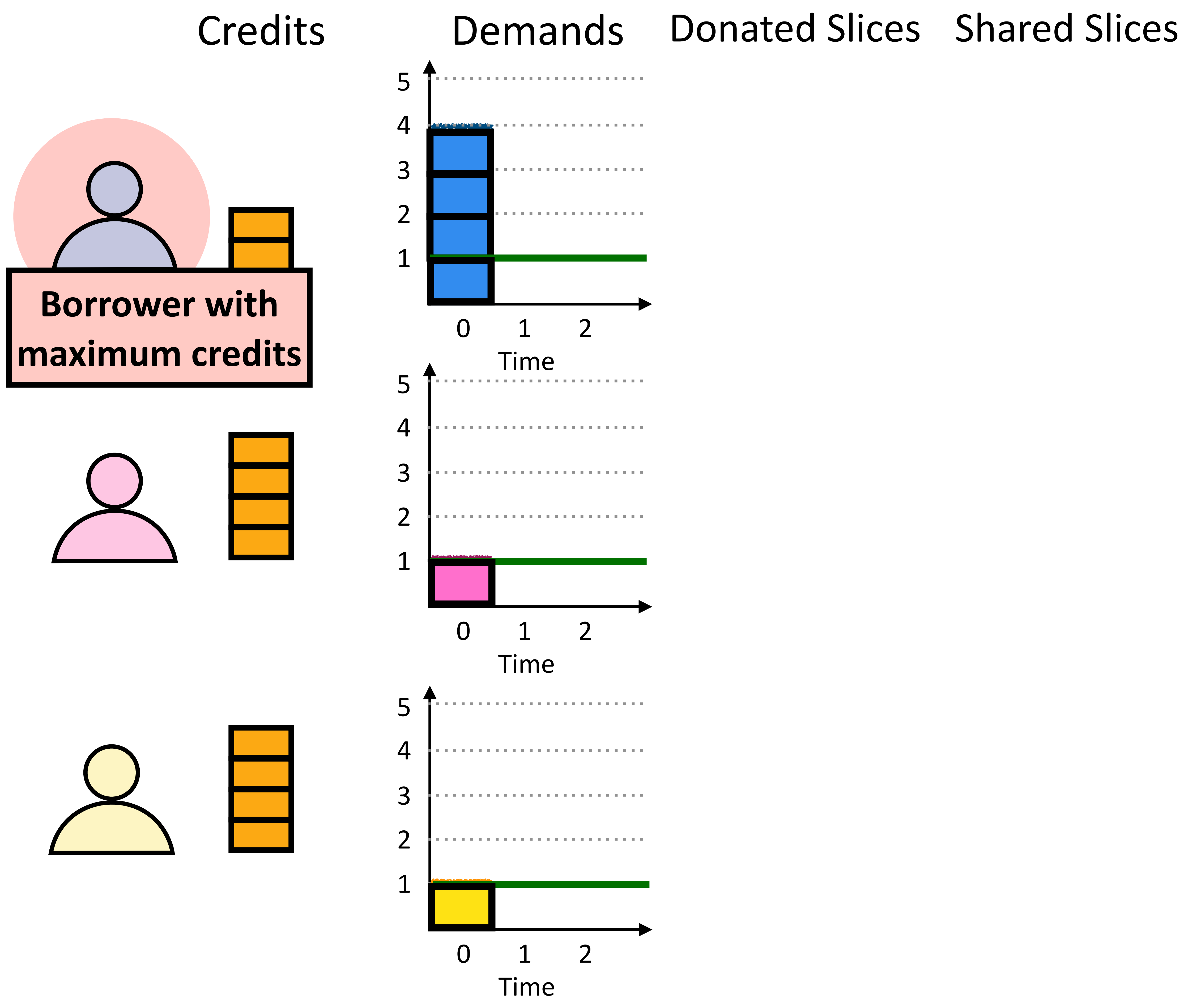
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

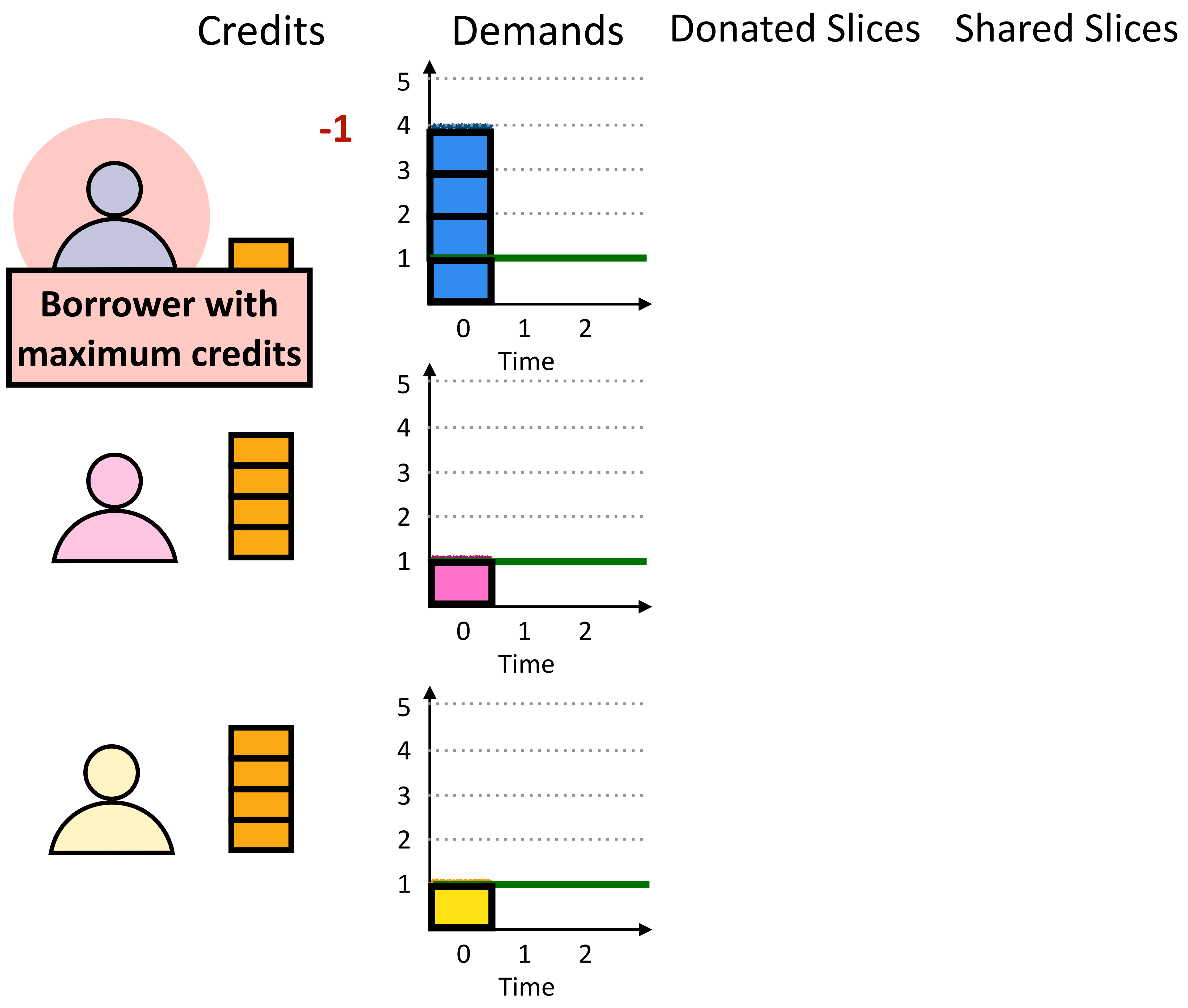
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

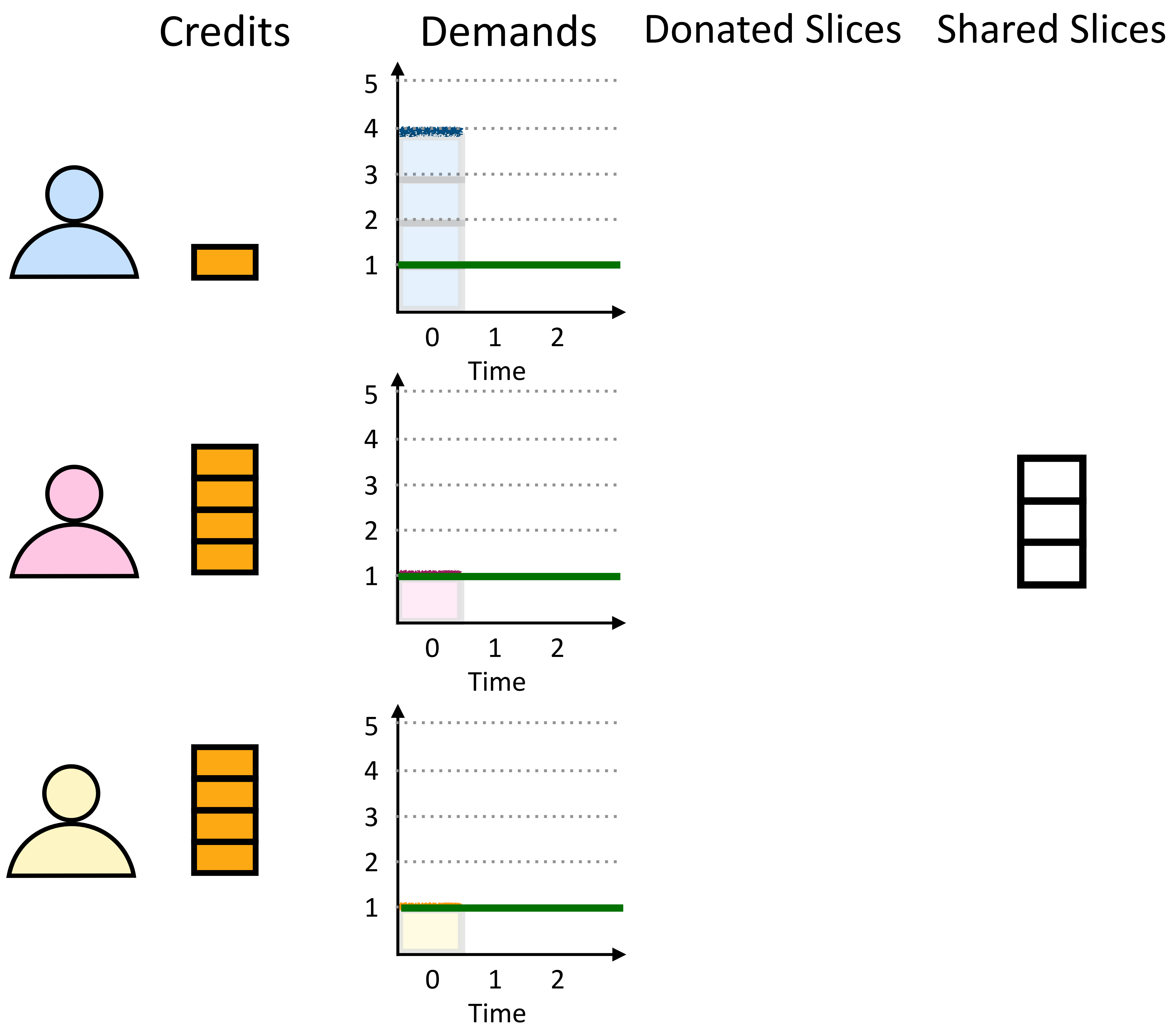
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

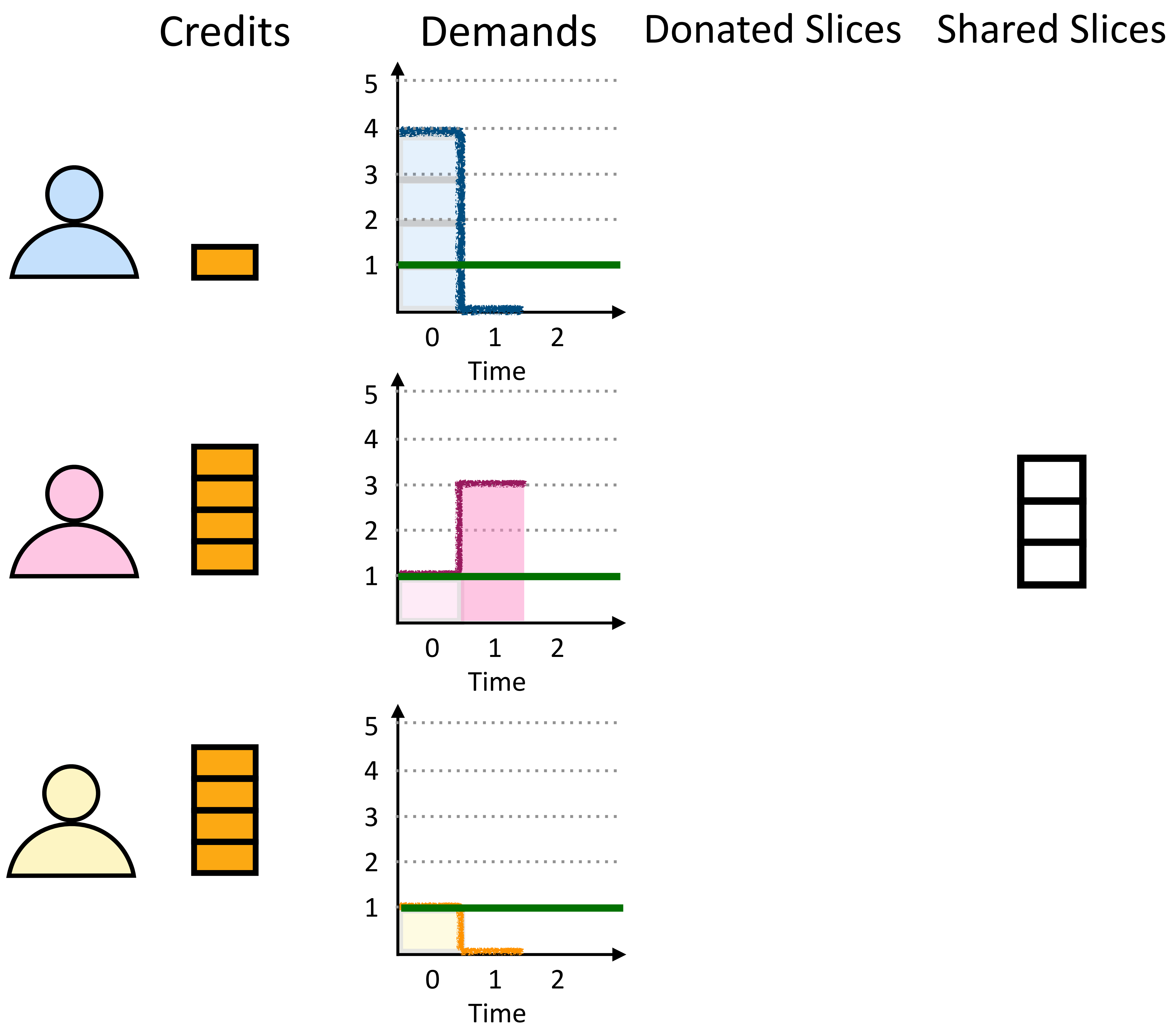
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

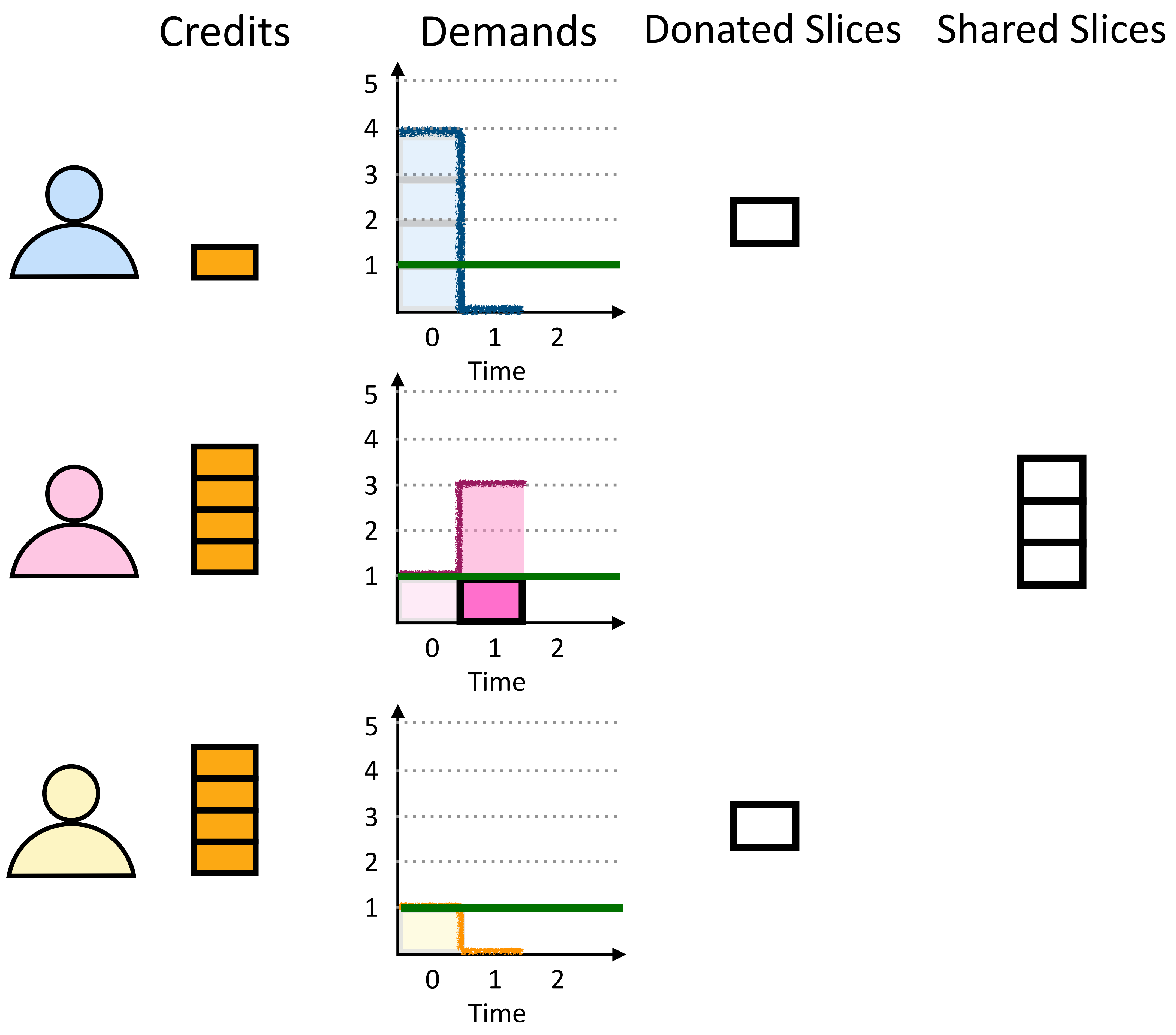
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

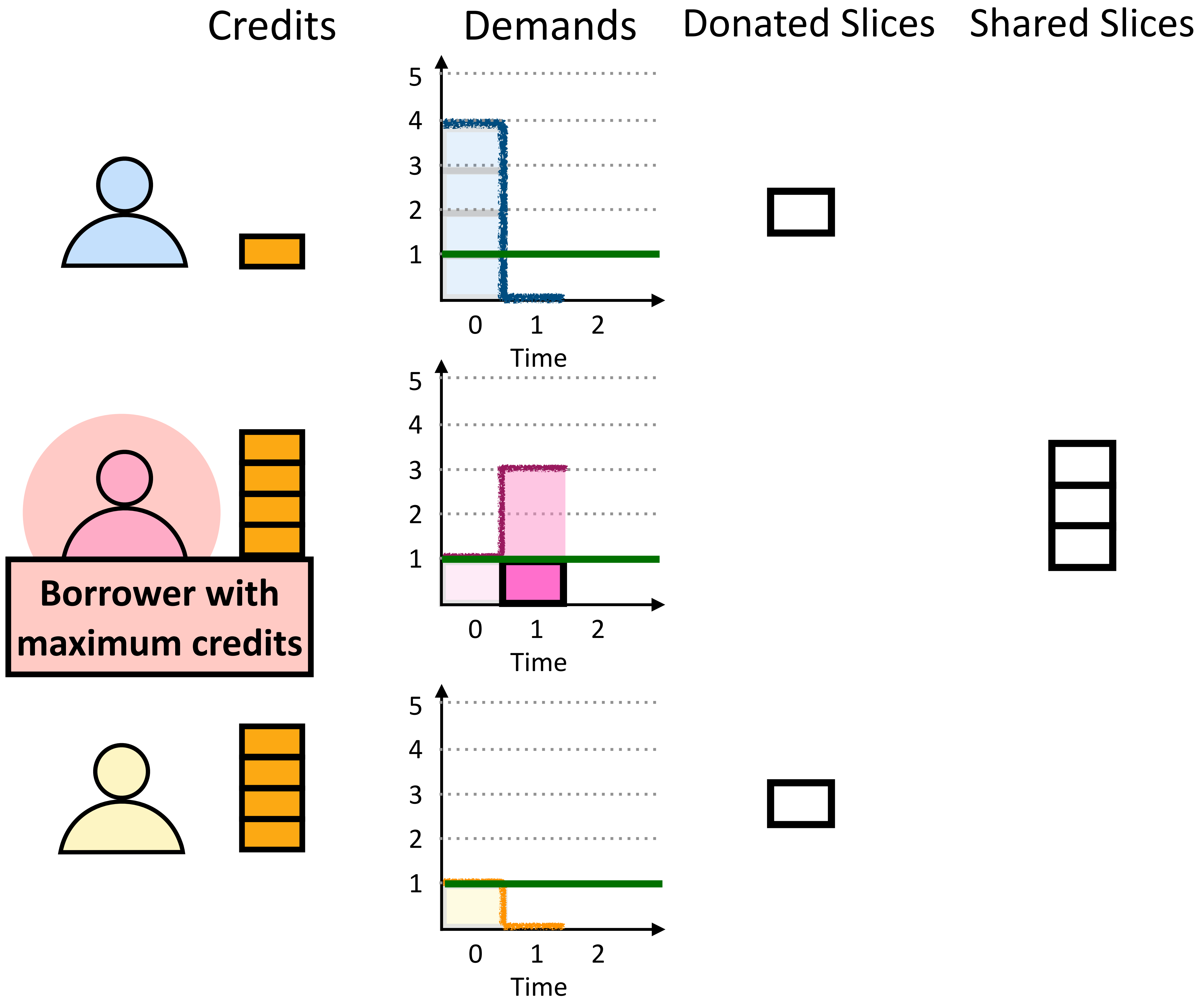
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

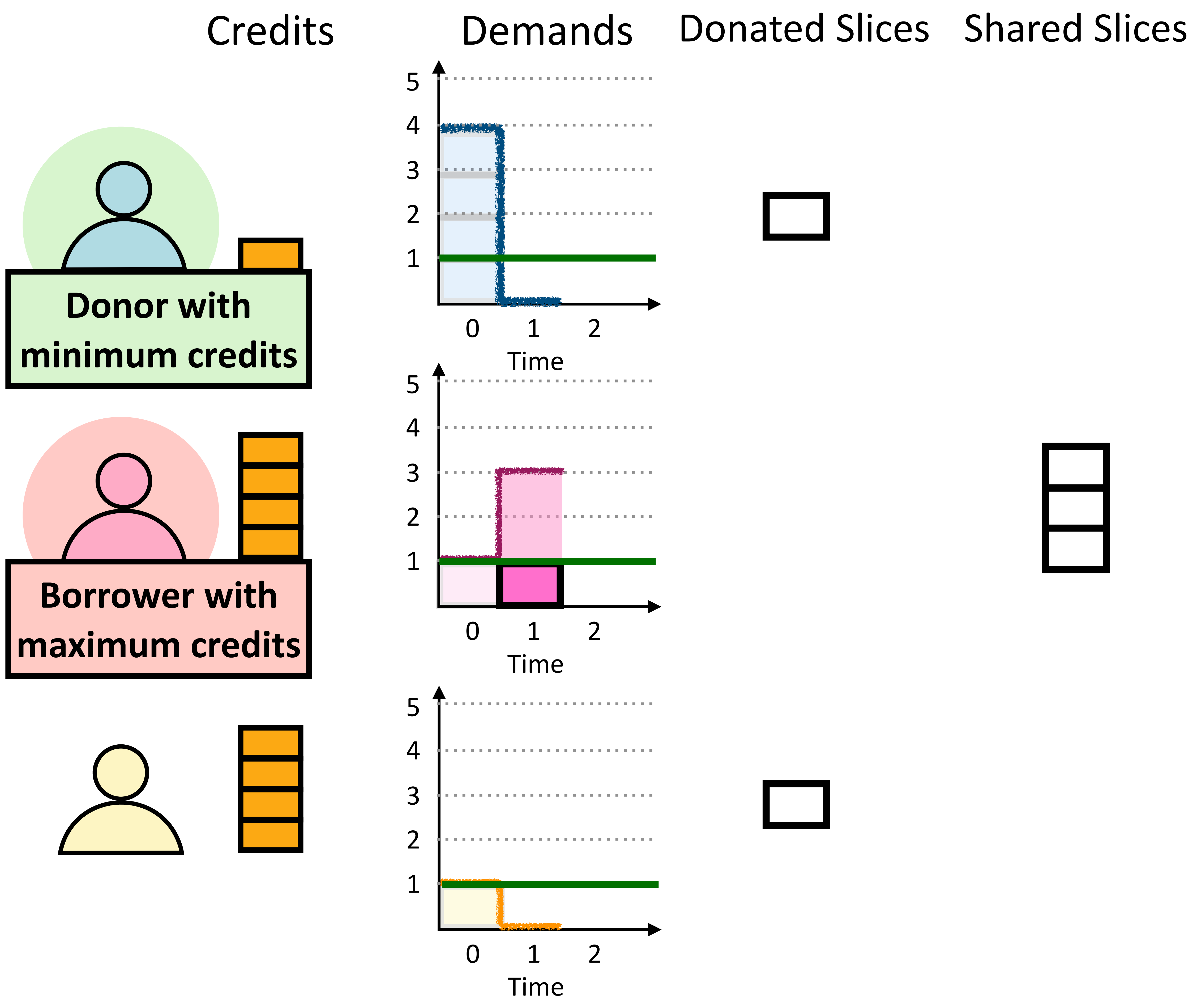
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

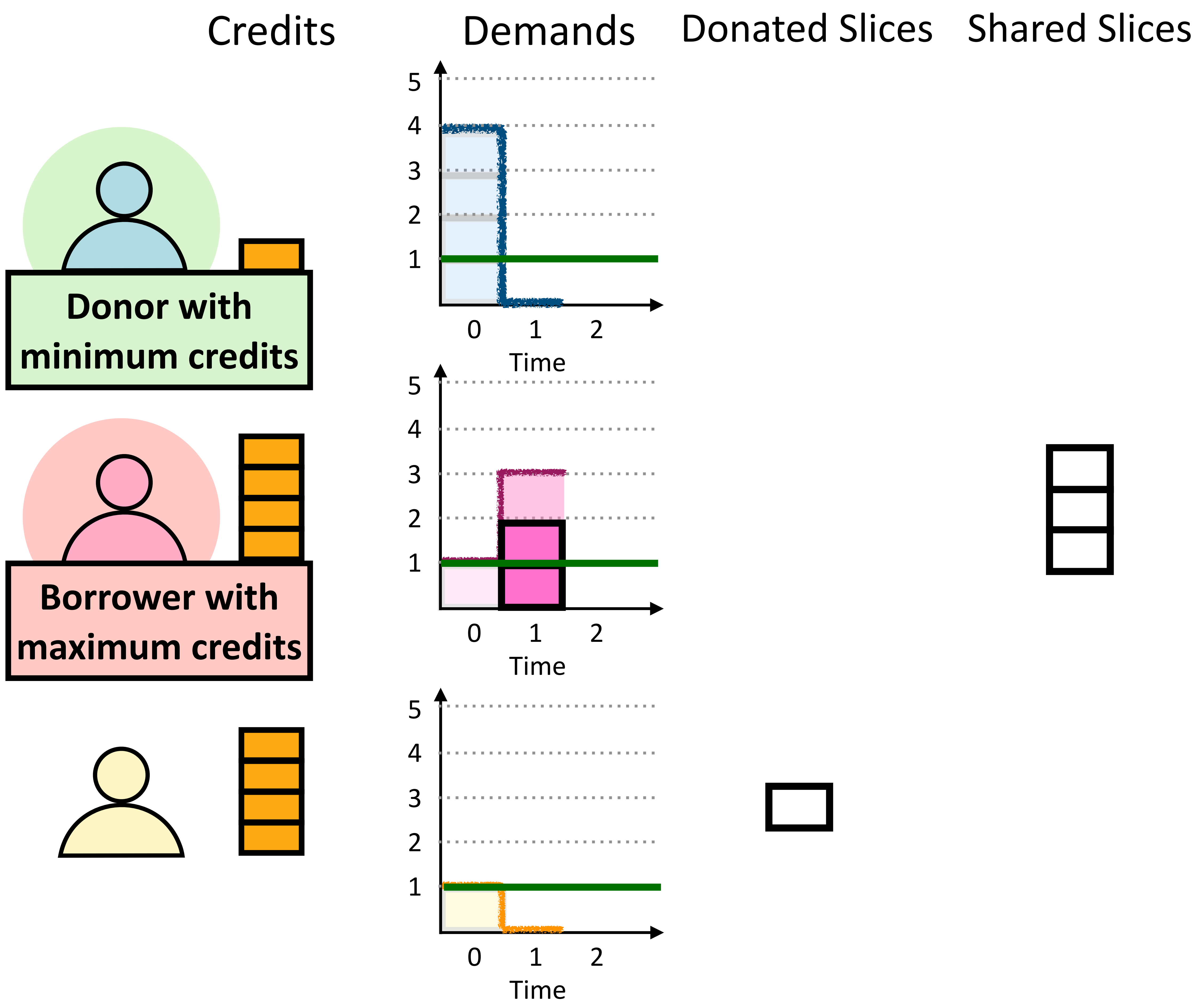
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

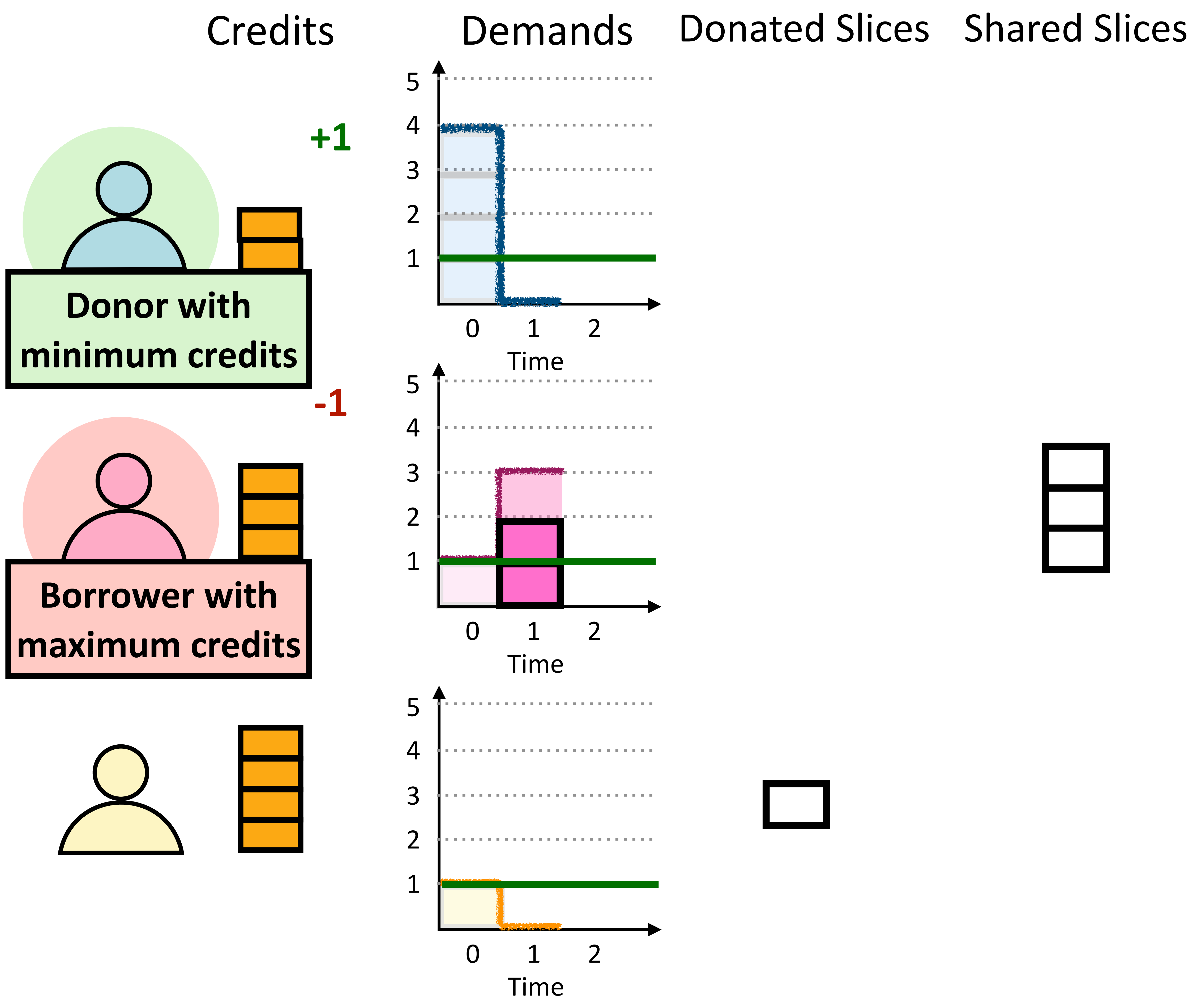
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

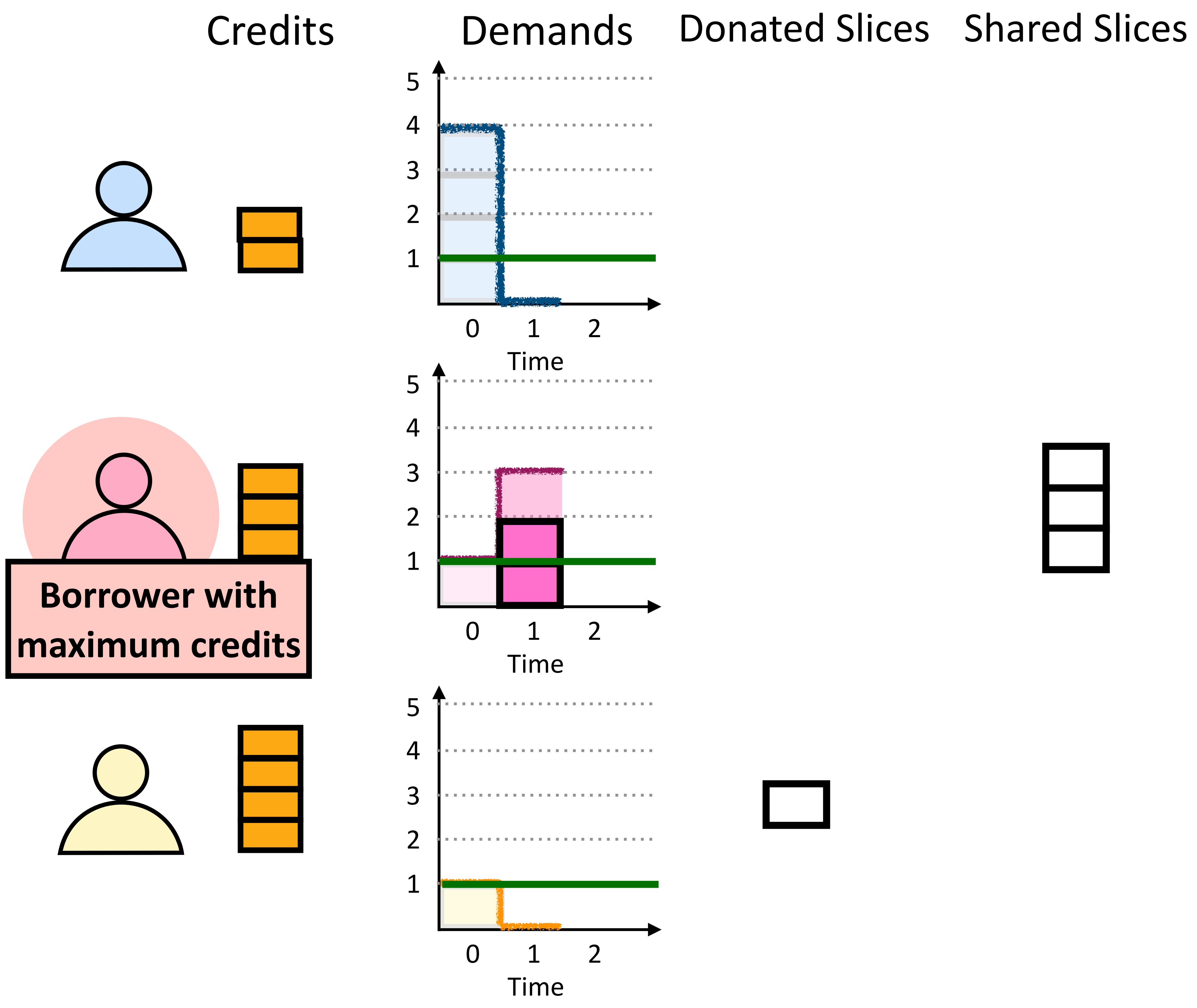
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

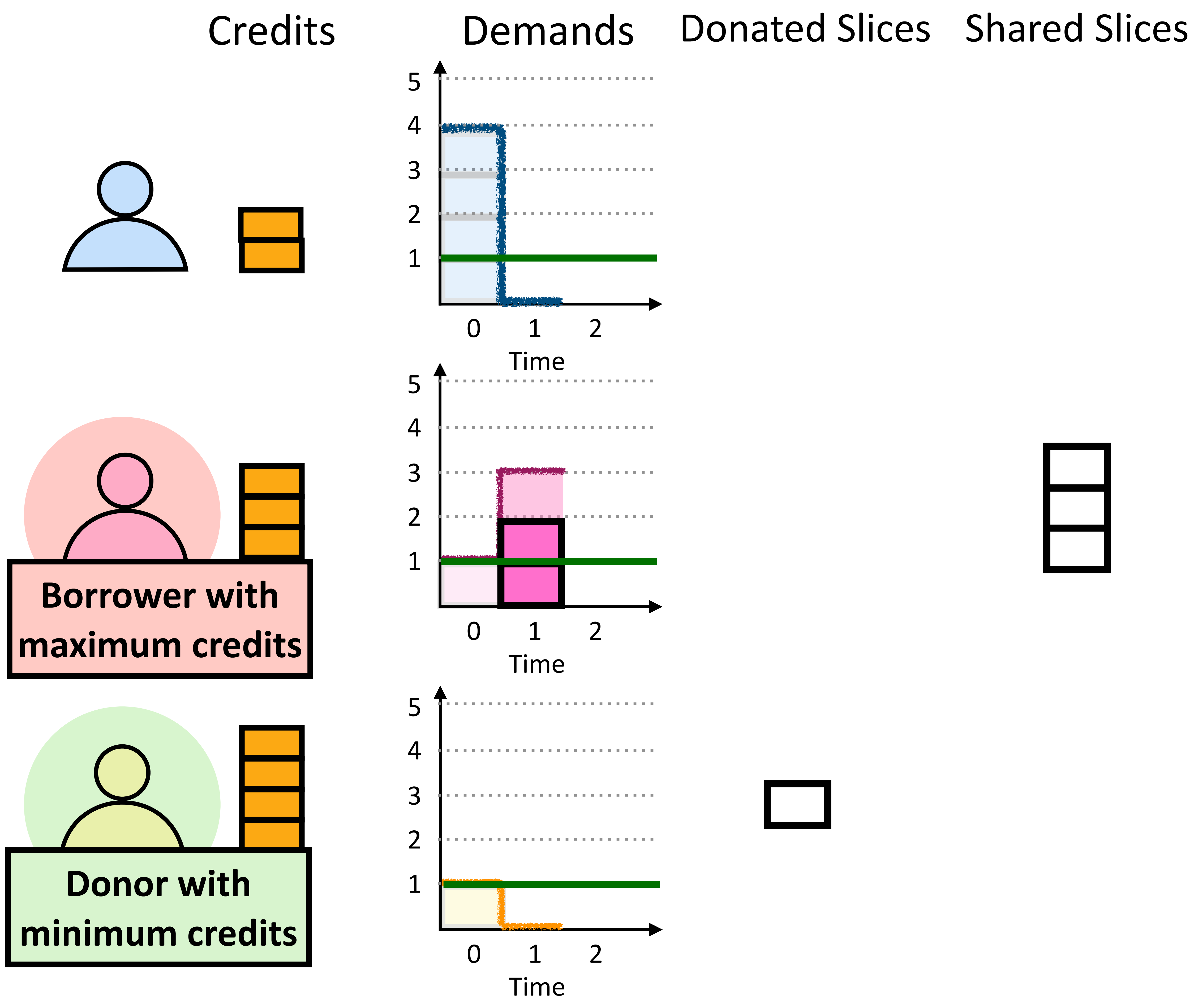
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

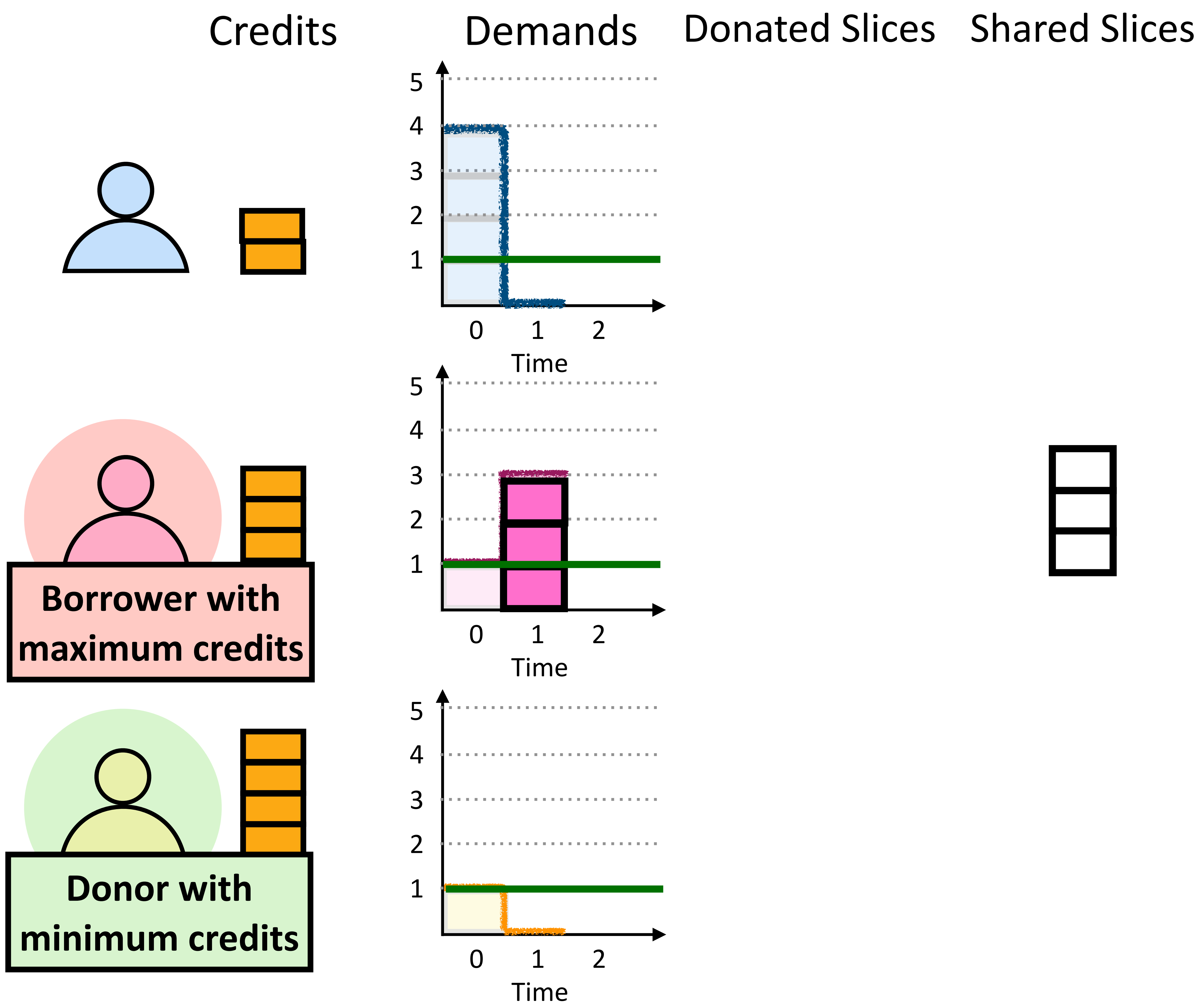
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

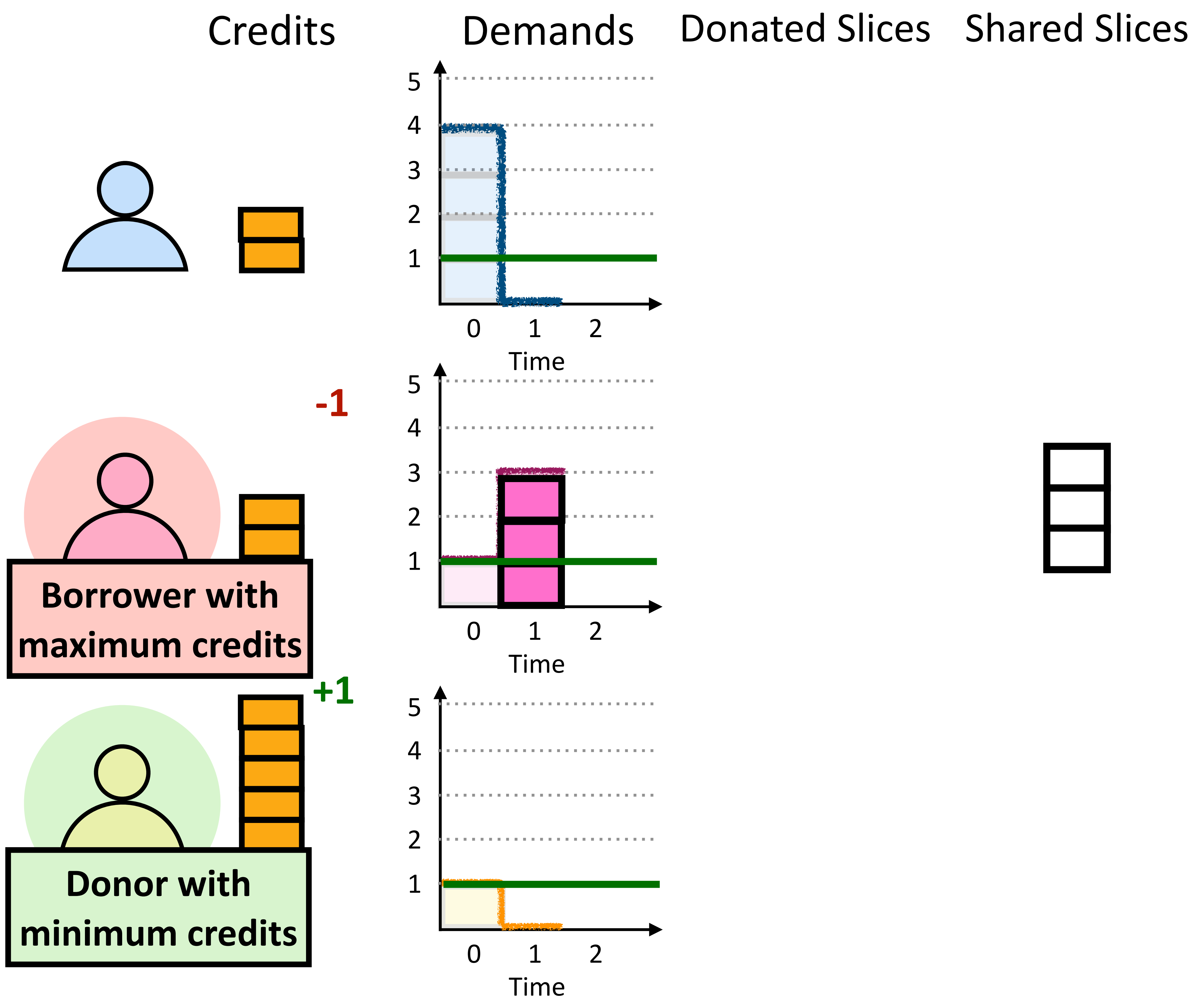
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

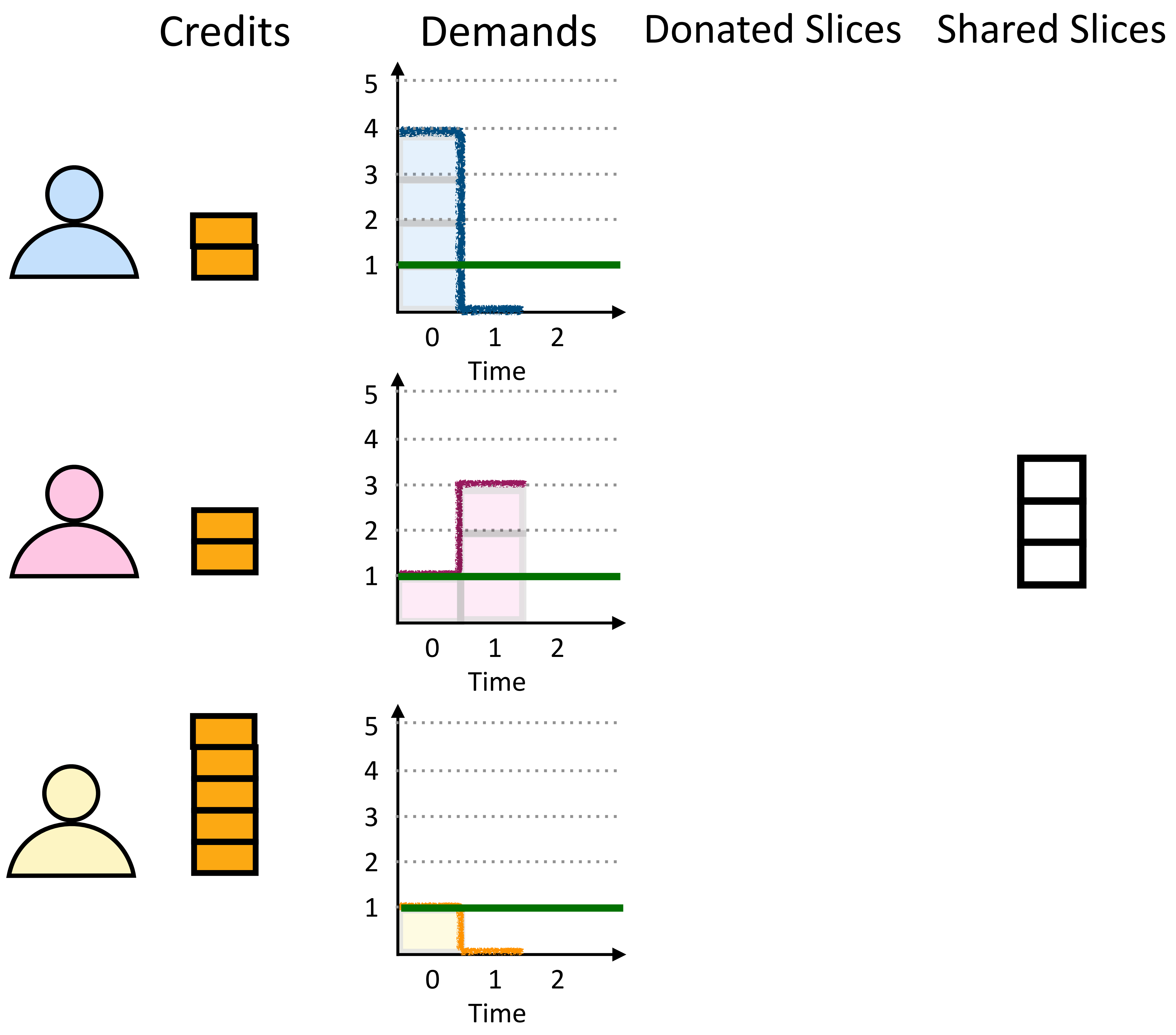
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

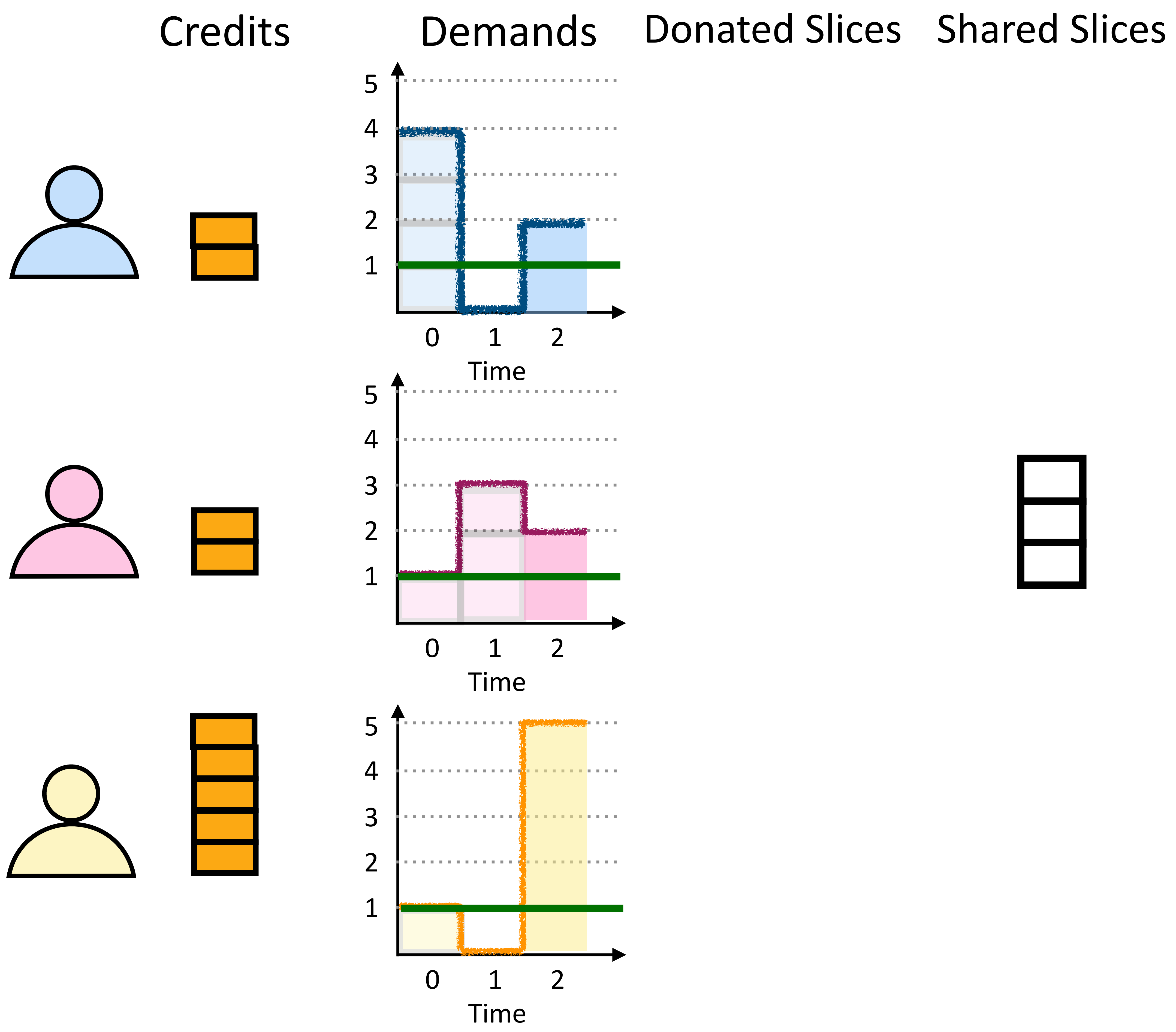
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

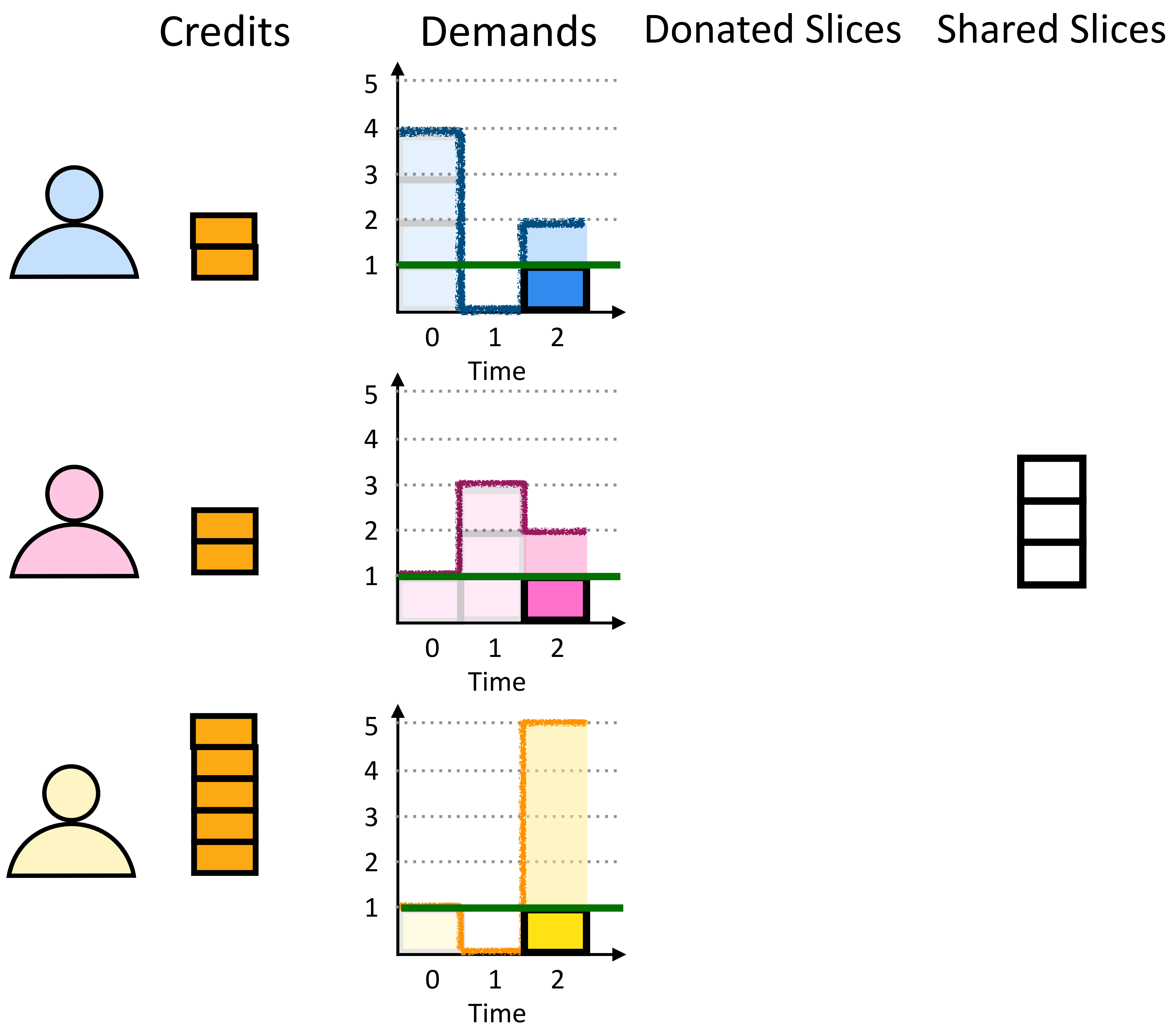
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

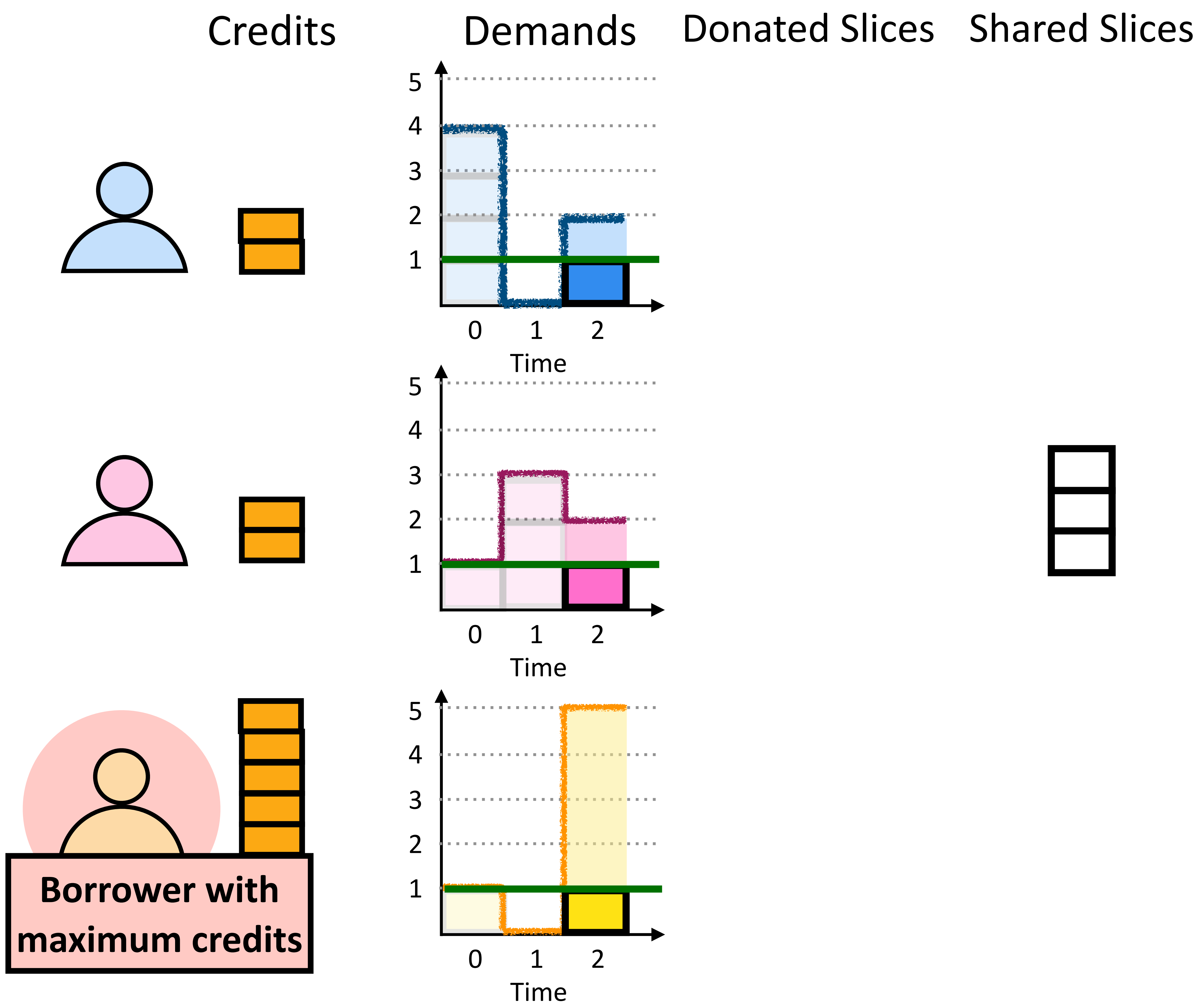
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

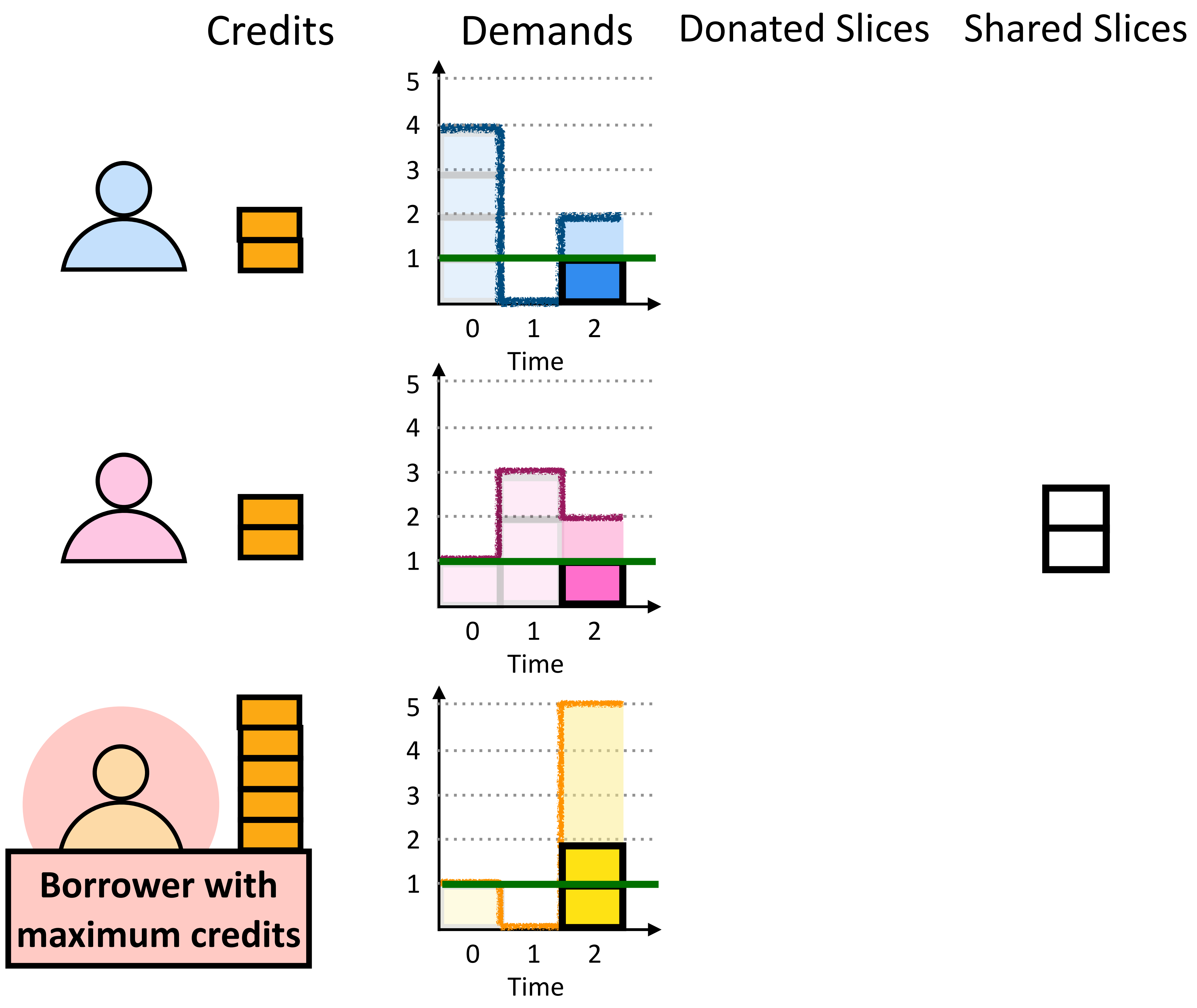
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

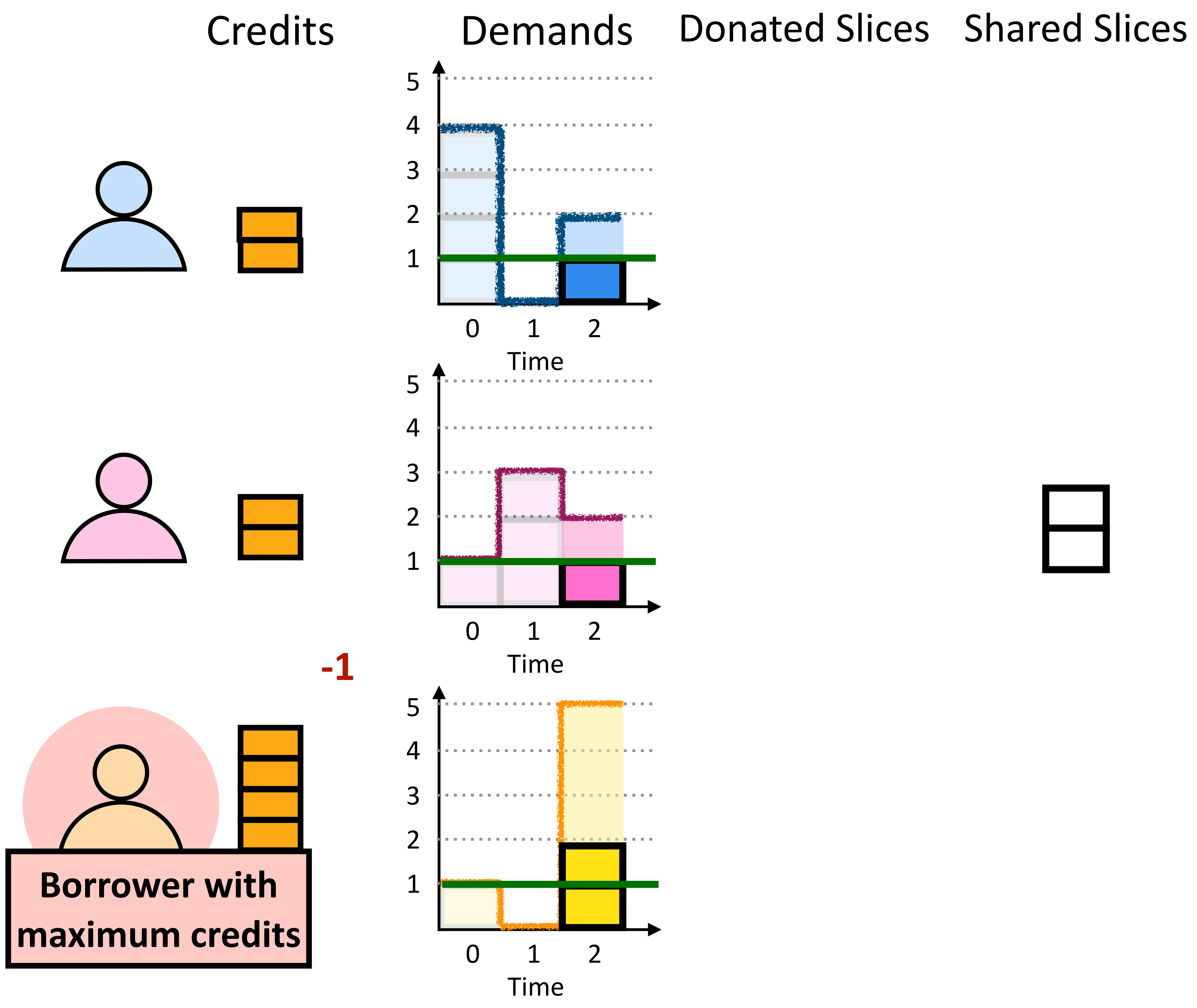
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

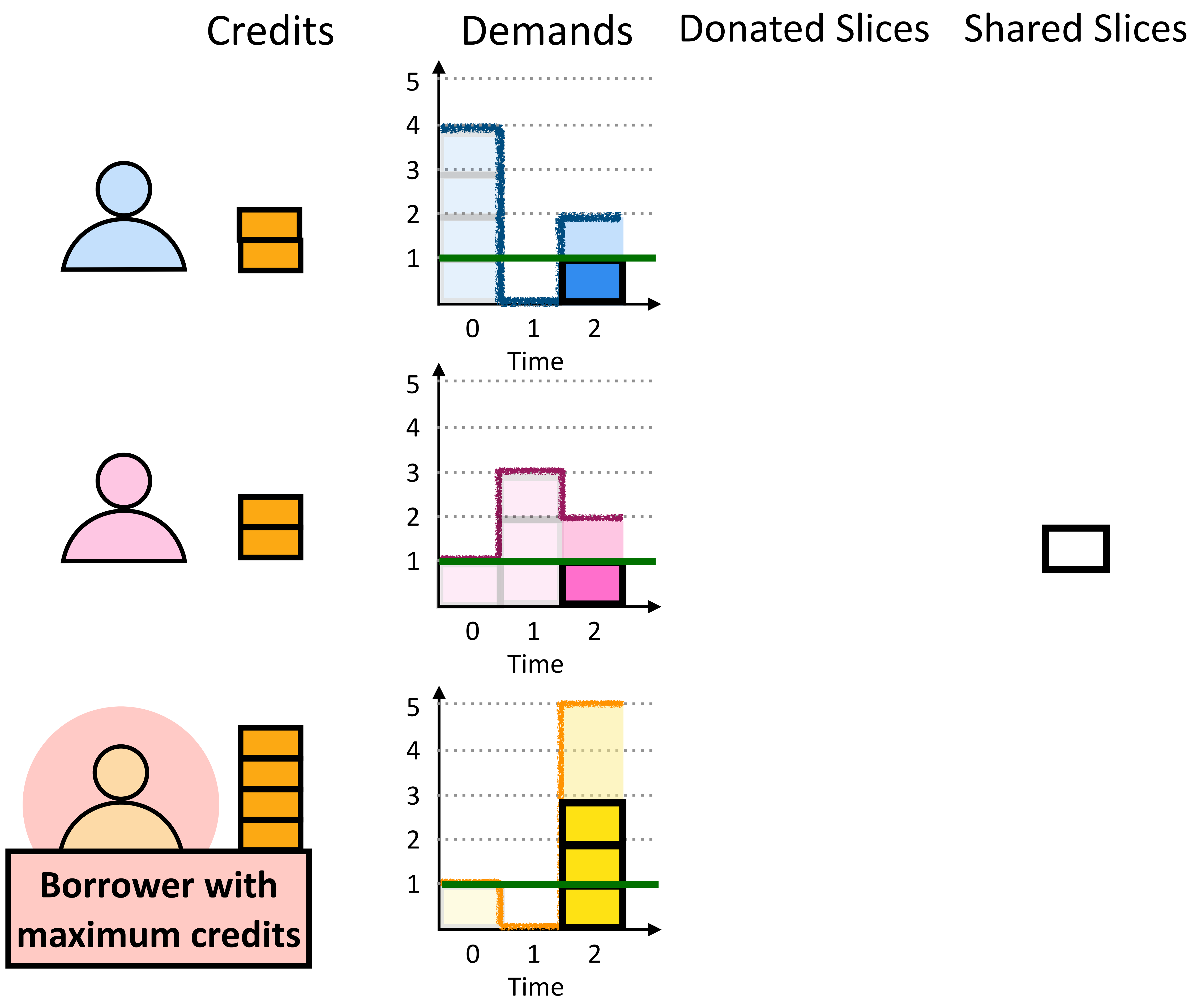
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

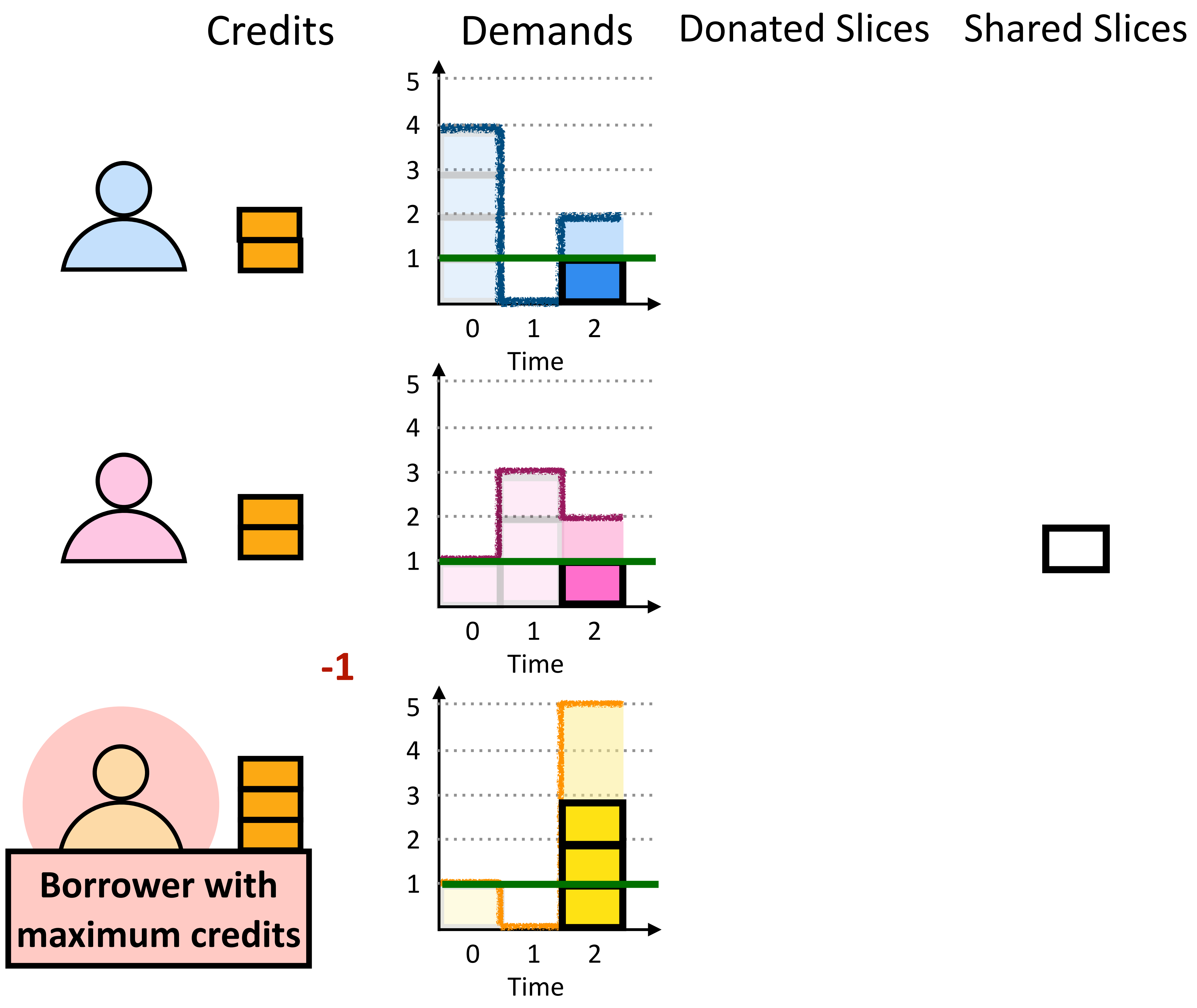
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits
If no donors, then use a shared slice

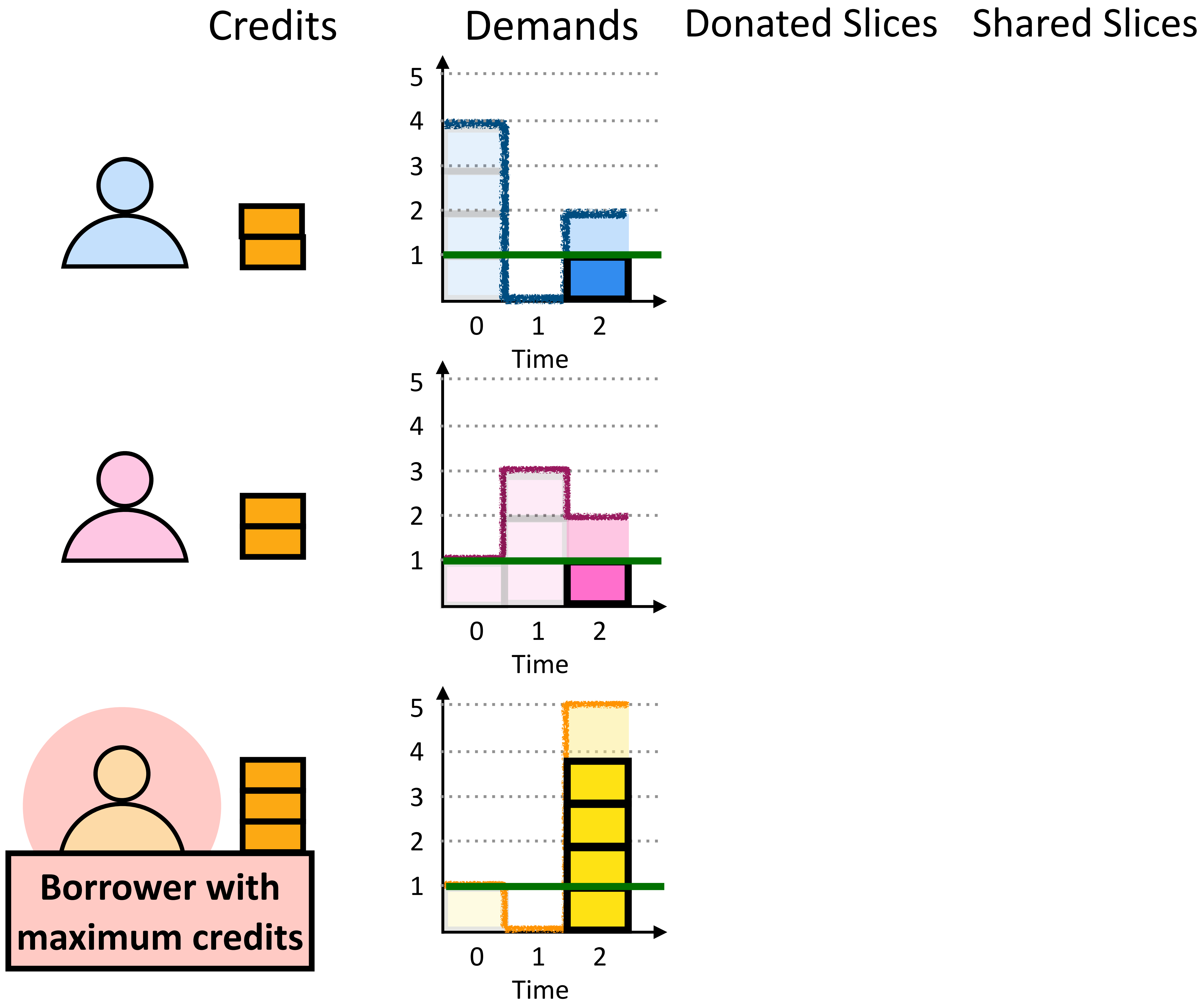
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

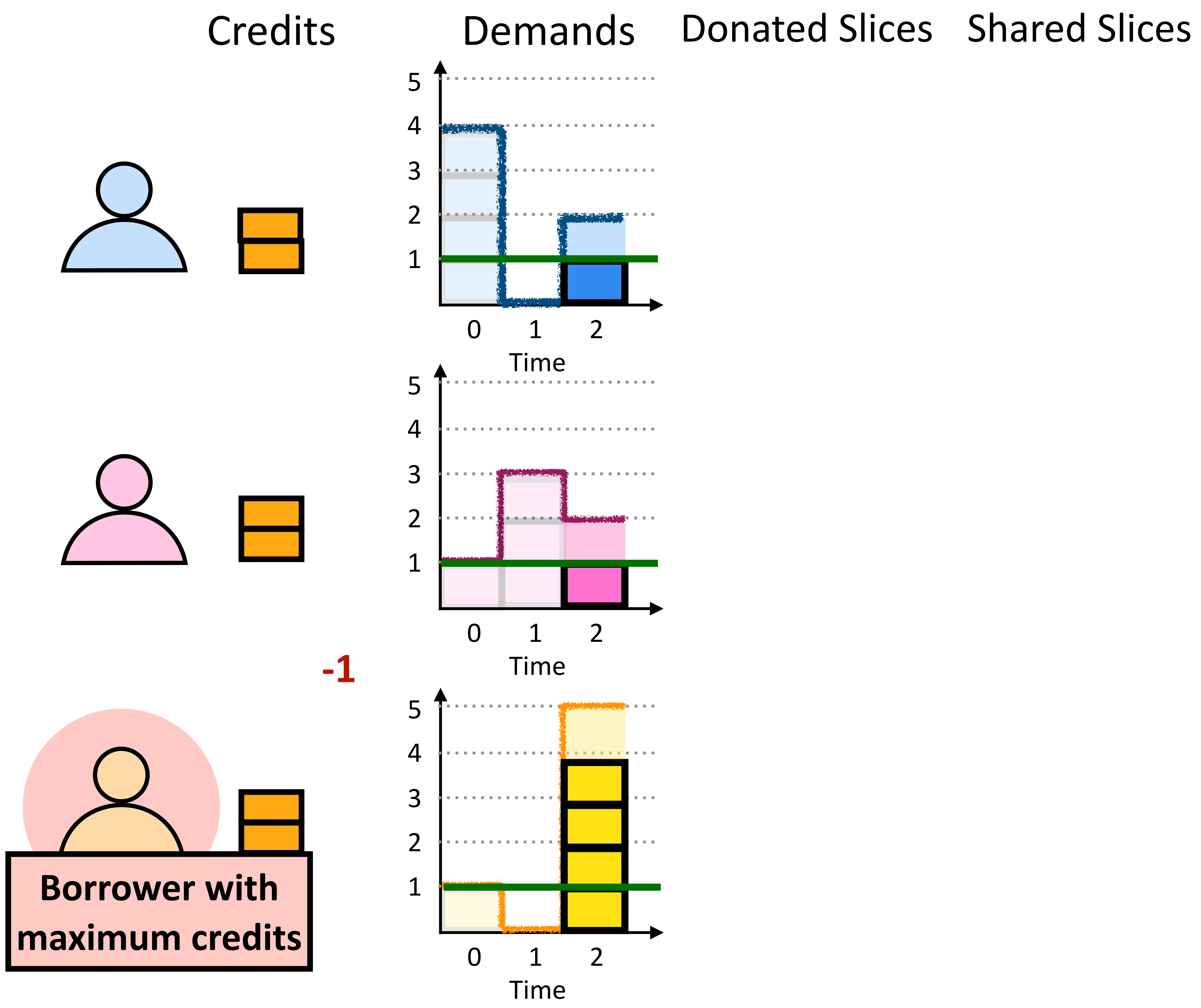
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

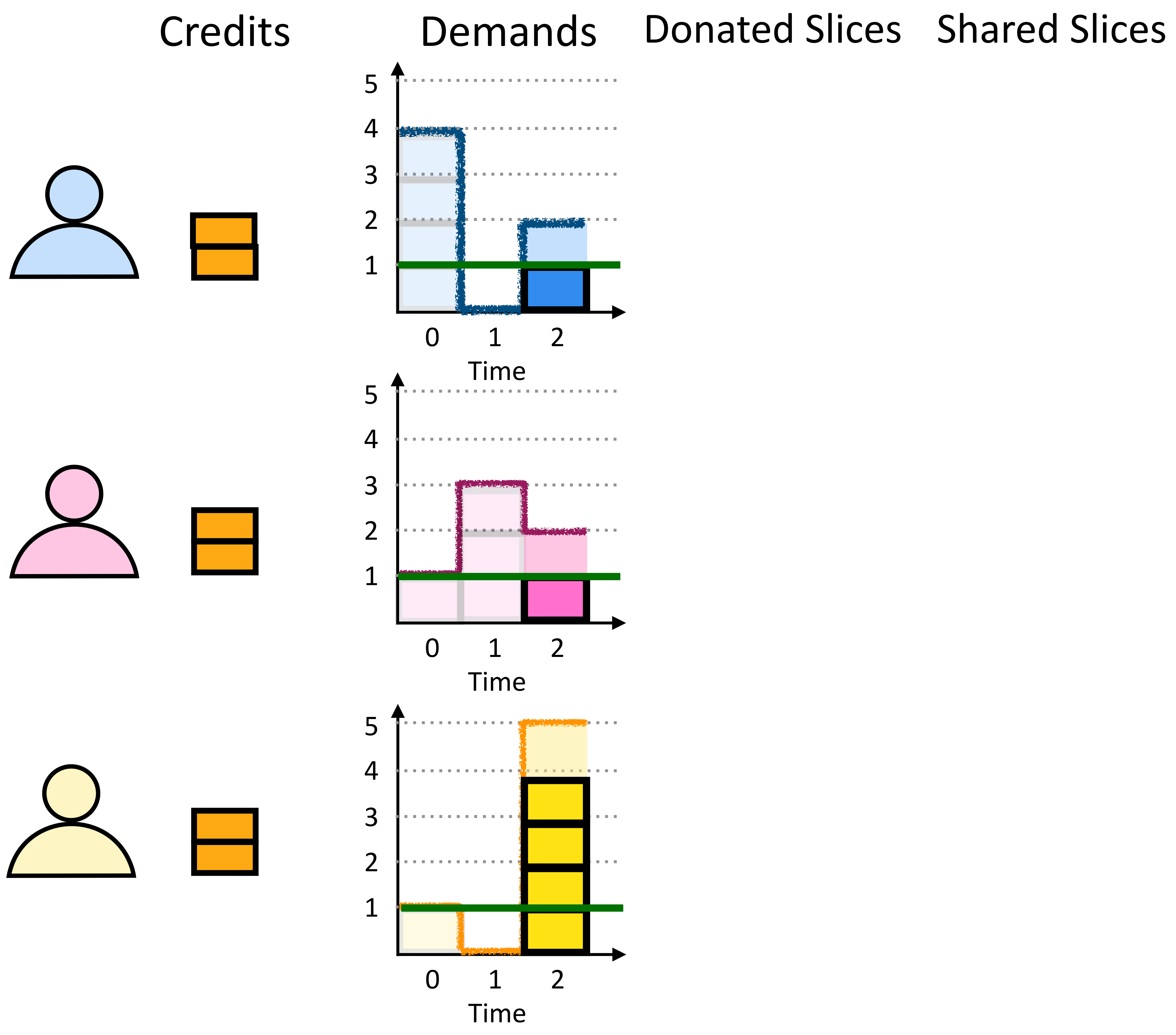
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

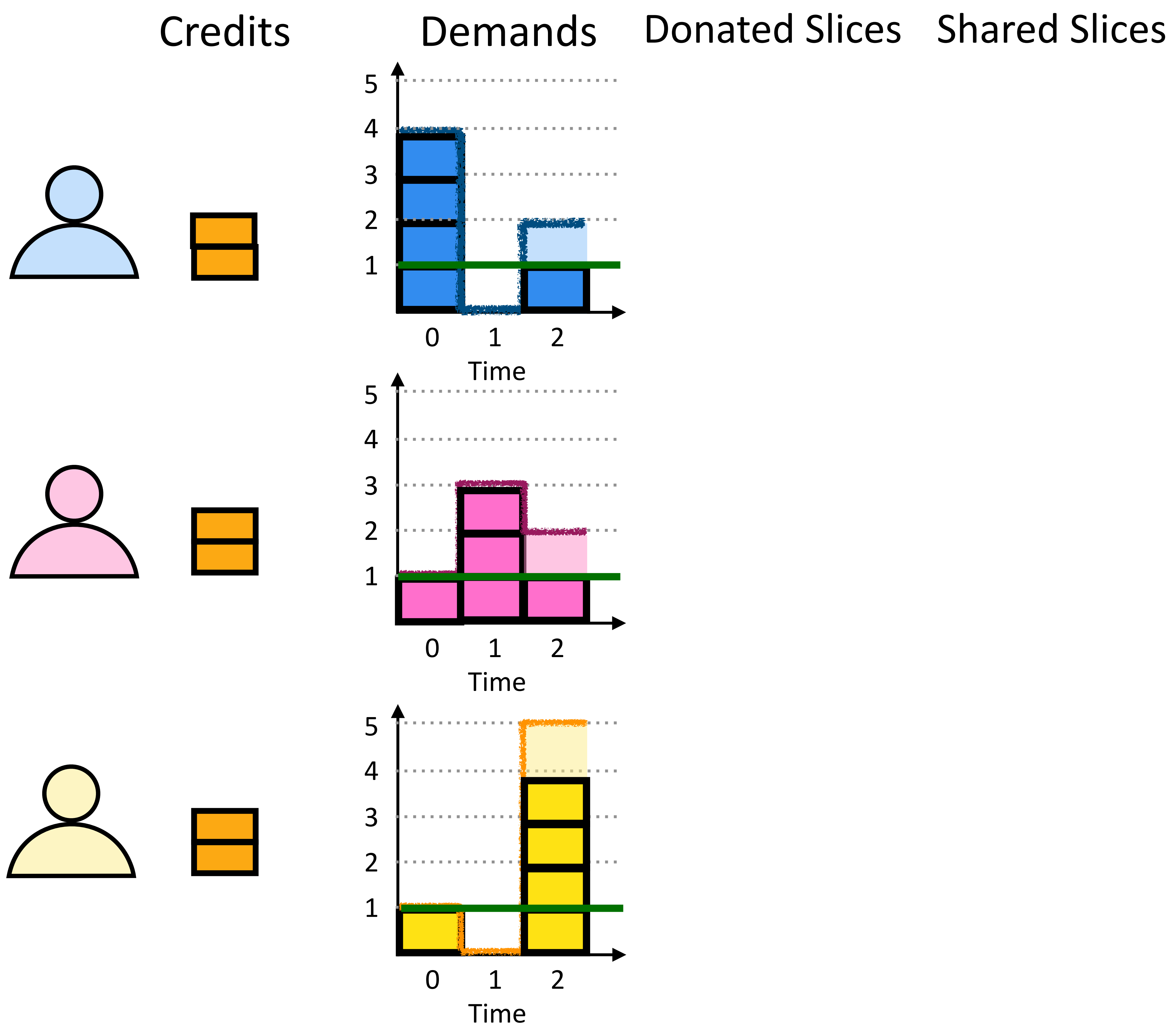
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma allocation algorithm

Running Example



Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

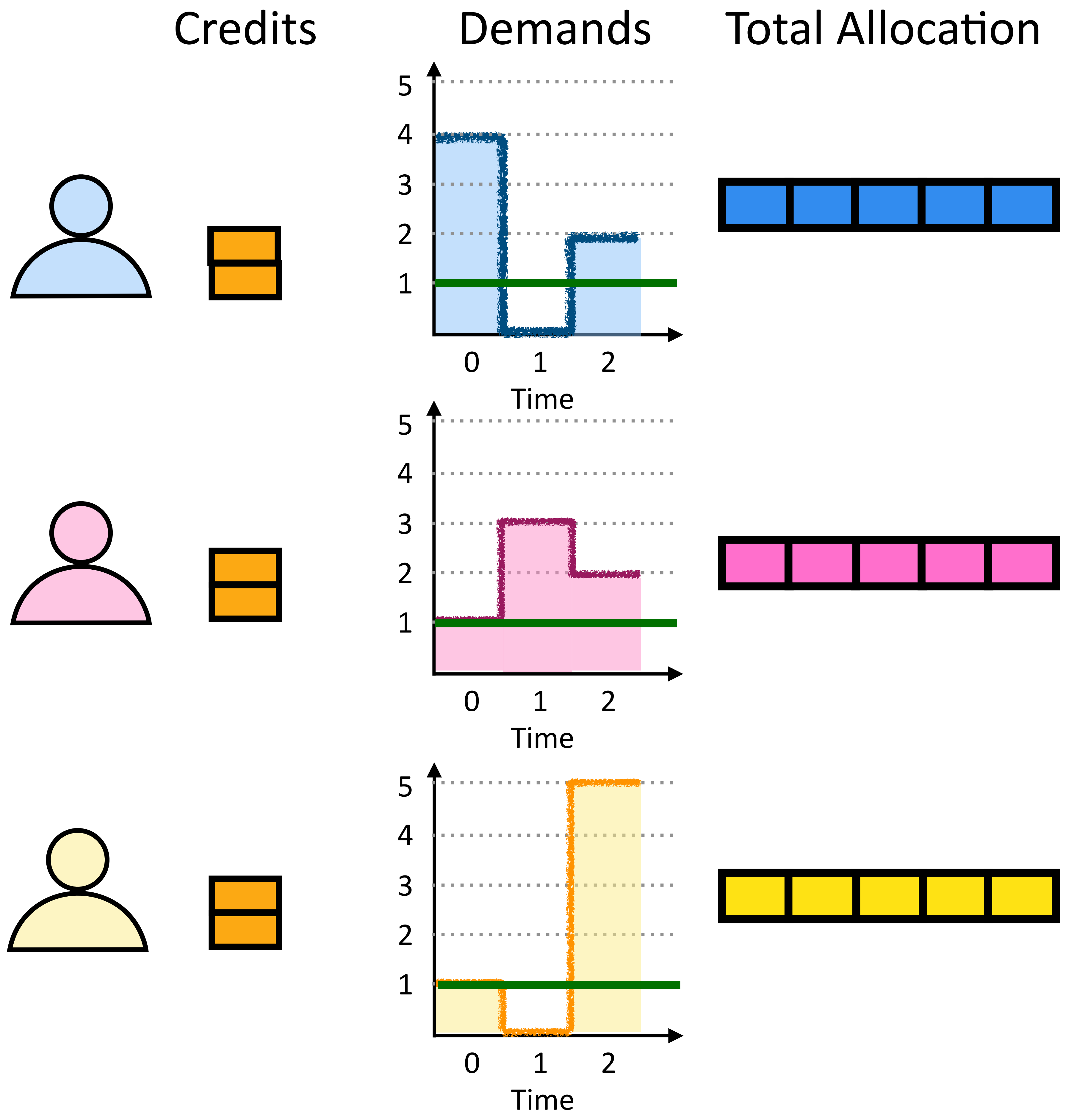
Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

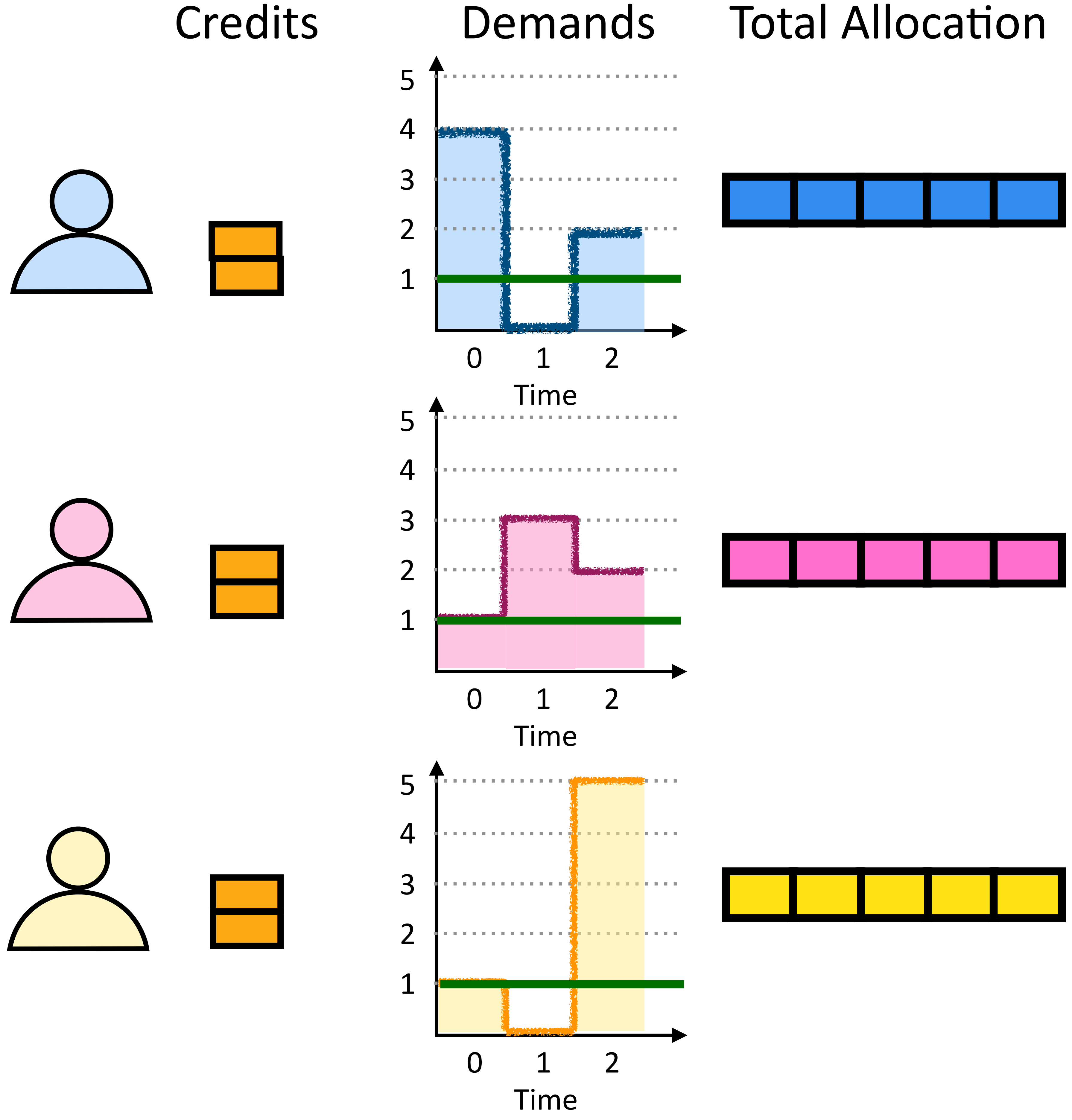
Karma allocation algorithm

Running Example



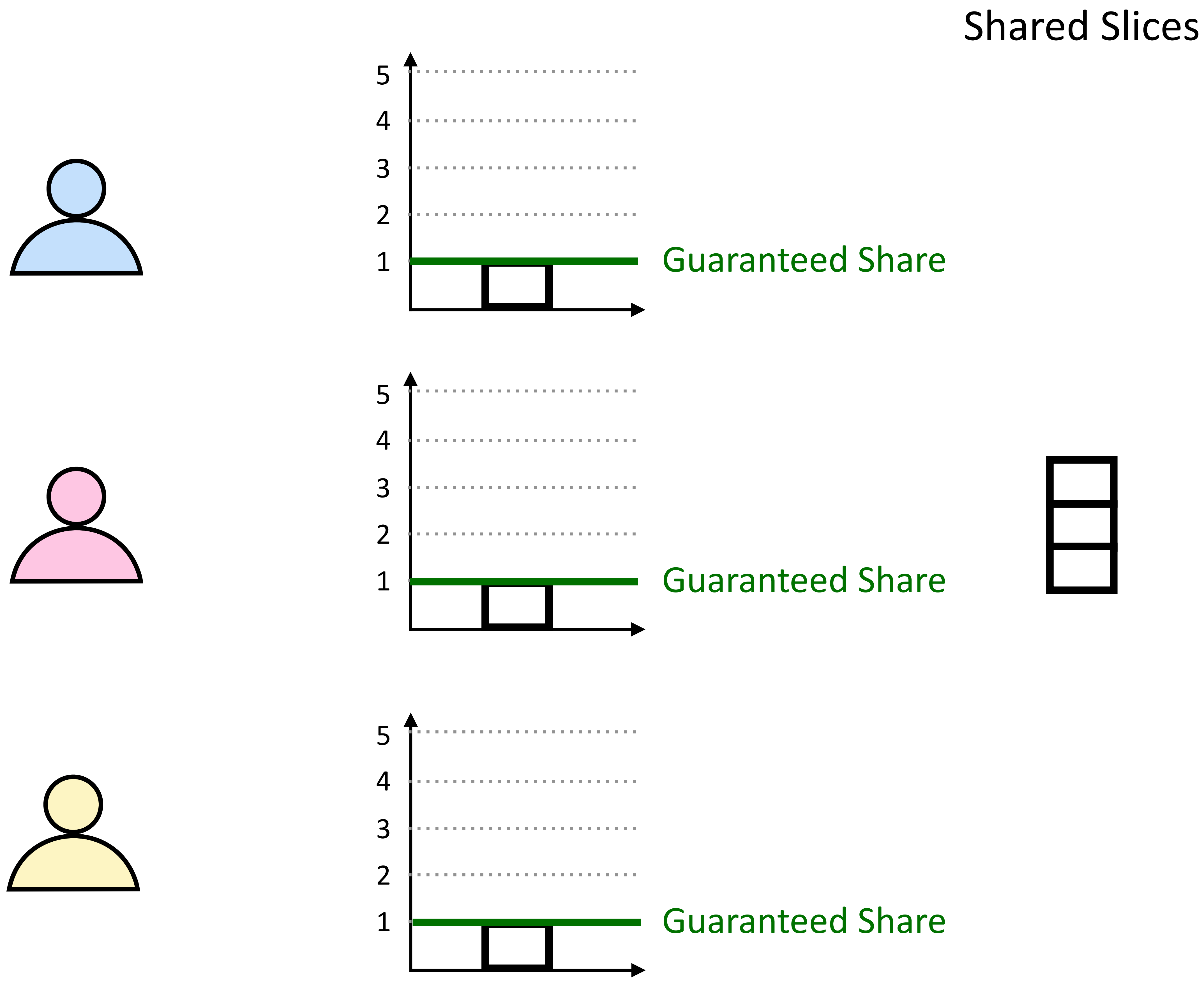
Karma allocation algorithm

Running Example

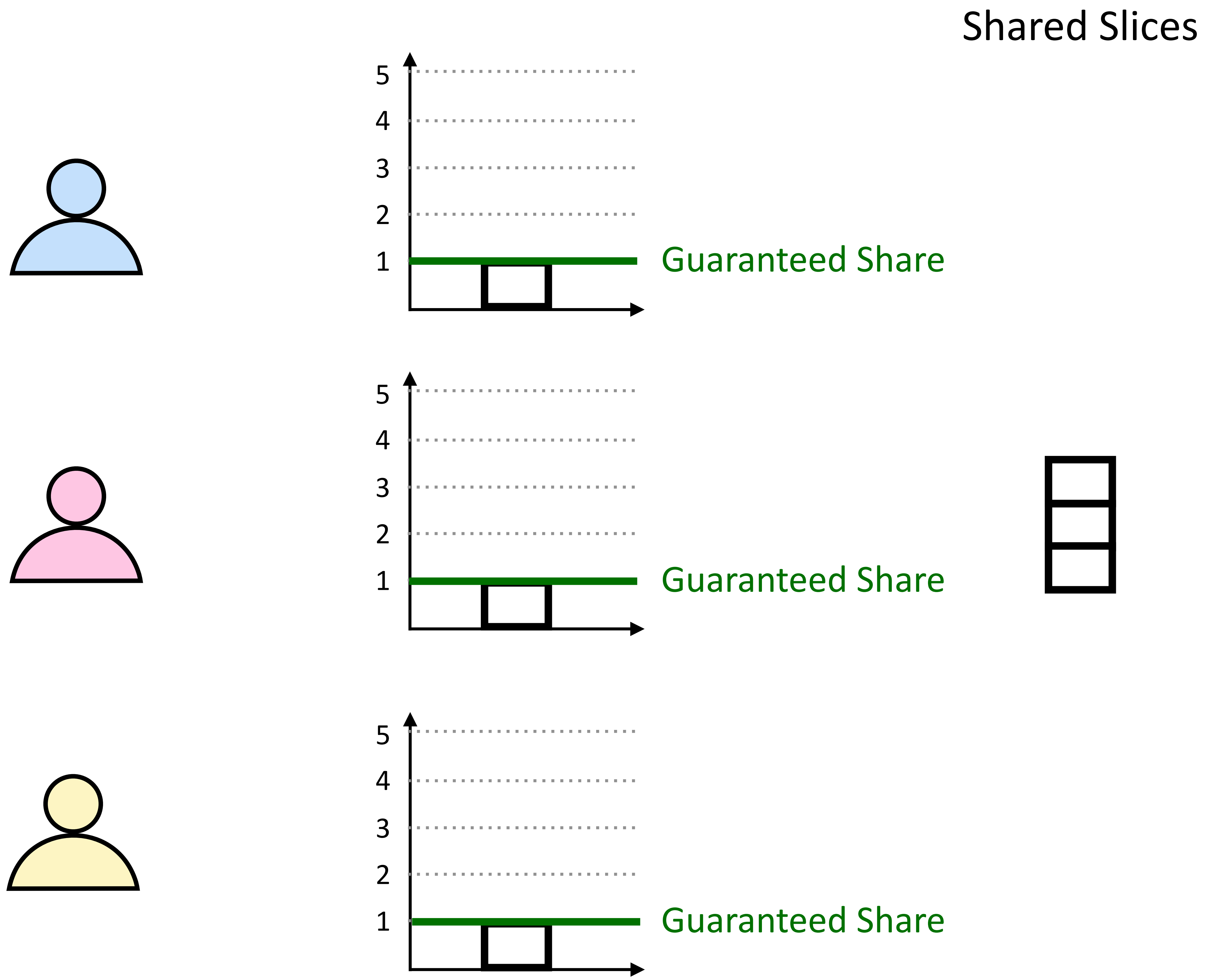


Equal total allocations!

Karma parameterizes the trade-off between instantaneous and long-term fairness

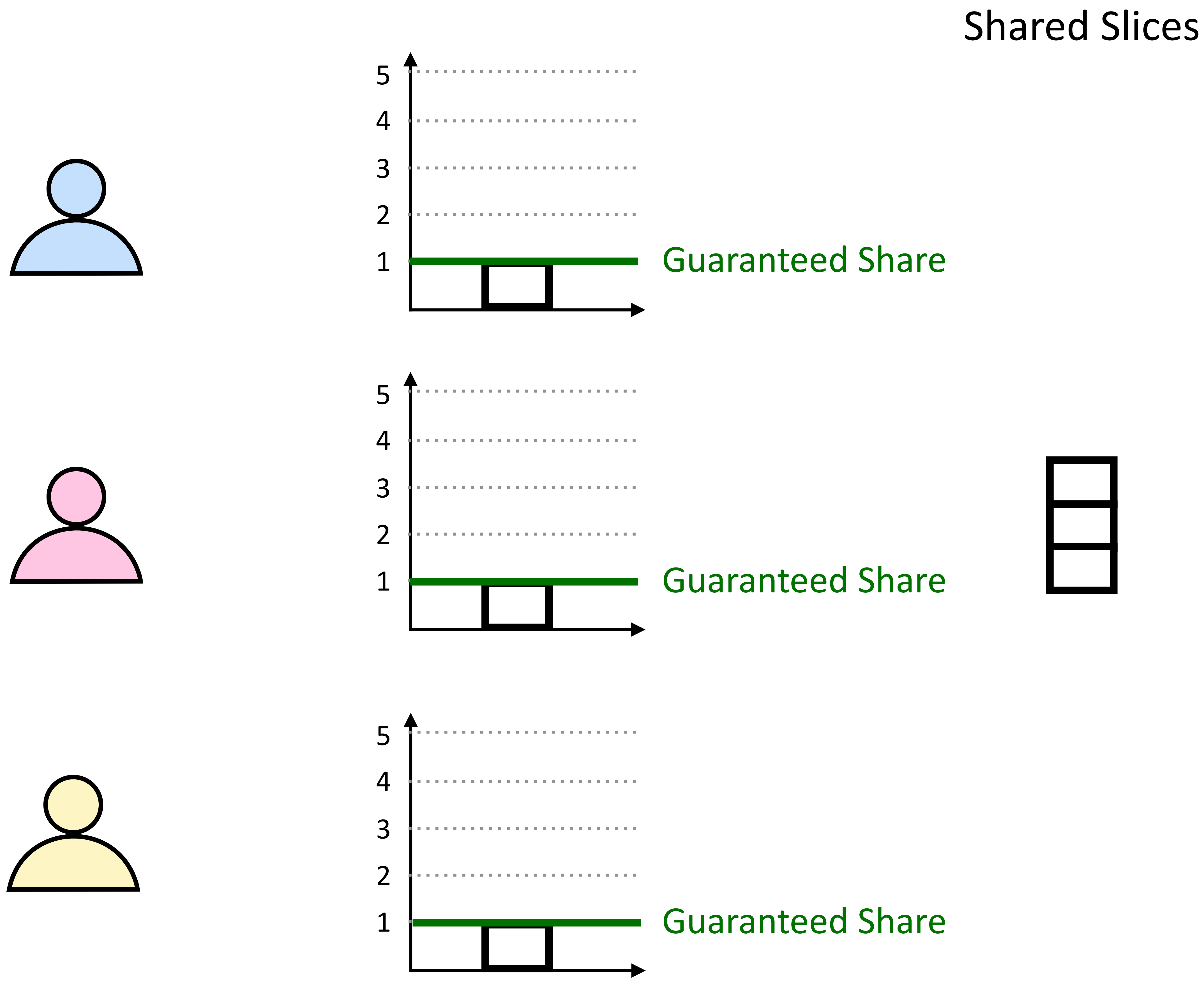


Karma parameterizes the trade-off between instantaneous and long-term fairness

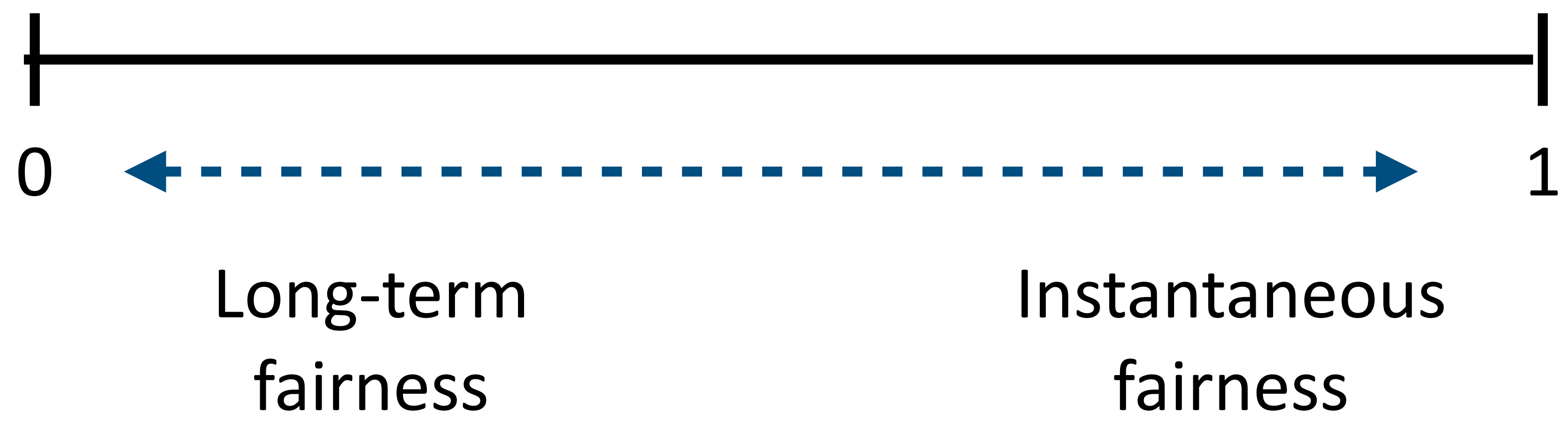


$$\frac{\text{Guaranteed share}}{\text{Fair share}}$$ is a parameter

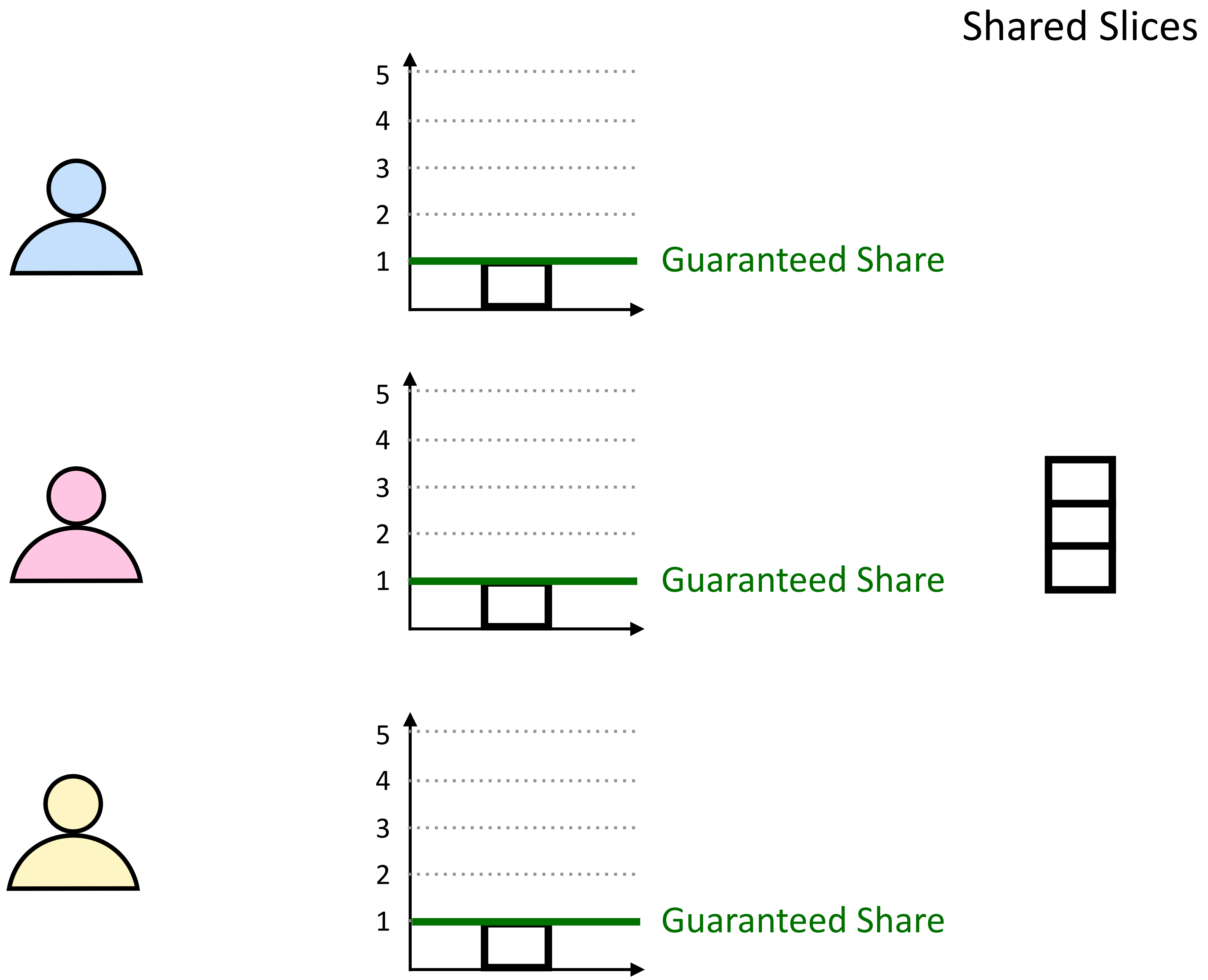
Karma parameterizes the trade-off between instantaneous and long-term fairness



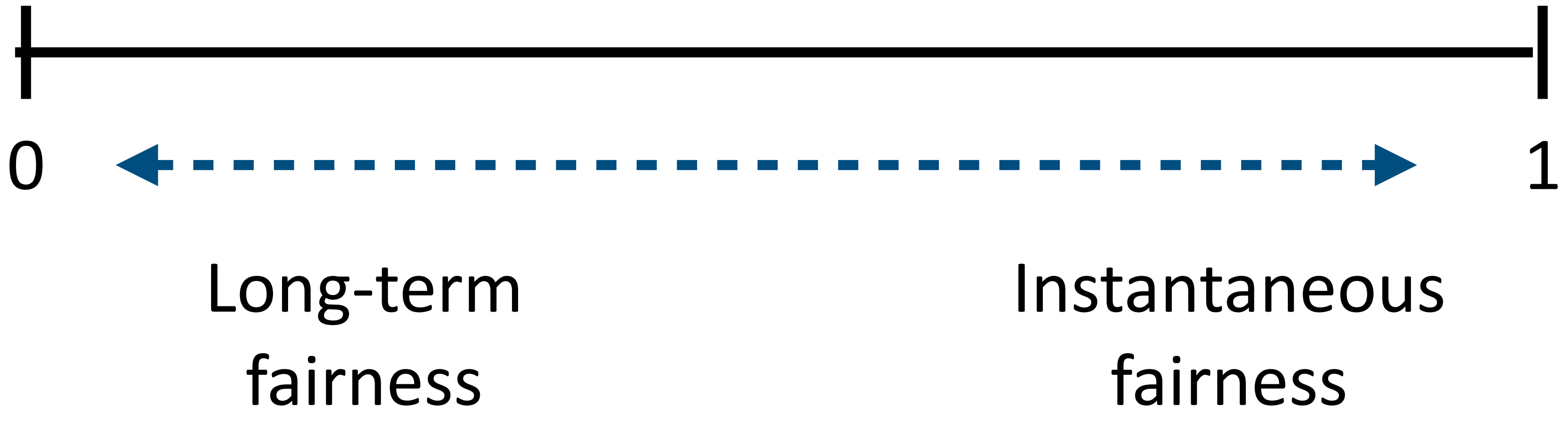
$$\frac{\text{Guaranteed share}}{\text{Fair share}}$$ is a parameter



Karma parameterizes the trade-off between instantaneous and long-term fairness



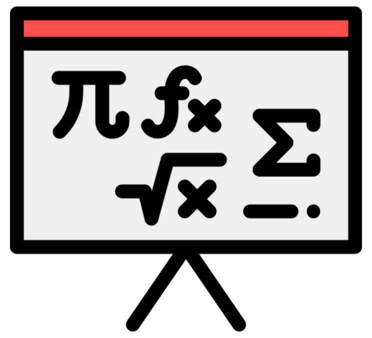
$$\frac{\text{Guaranteed share}}{\text{Fair share}} \text{ is a parameter}$$



(See paper for more details)

Karma: Revisiting the classical resource allocation problem under dynamic demands

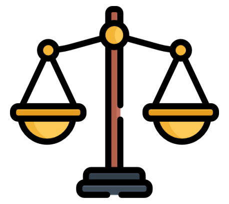
New resource allocation algorithm for dynamic demands



Theoretical guarantees

Pareto efficiency

Strategy-proofness



Fairness

Prototype implementation and evaluation

Karma is Pareto efficient

Pareto efficiency

Resources should not remain unused when there is demand

Karma is Pareto efficient

Pareto efficiency

Resources should not remain unused when there is demand

Karma Allocation Algorithm

Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

Allocate slice to borrower

-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma is Pareto efficient

Pareto efficiency

Resources should not remain unused when there is demand

Karma Allocation Algorithm

Pick **borrower** with **maximum** credits

Pick **donor** with **minimum** credits

If no donors, then use a shared slice

Allocate slice to borrower

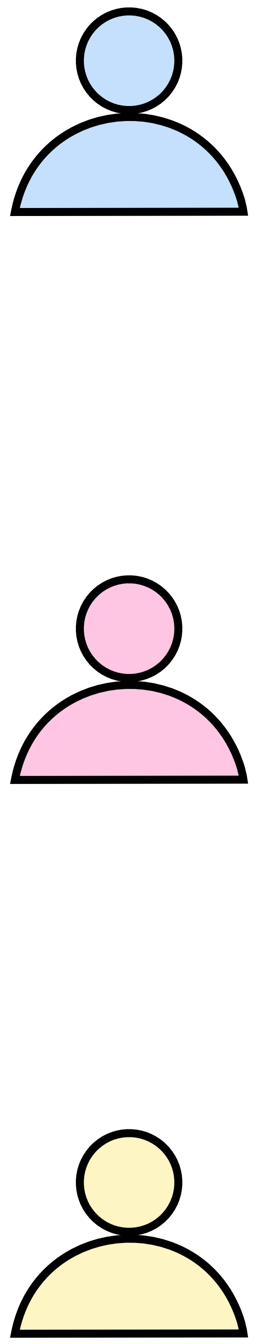
-1 credit for borrower, +1 credit for donor

(Repeat until demands satisfied or resources exhausted)

Karma provides powerful strategy-proofness properties

Strategy-proofness

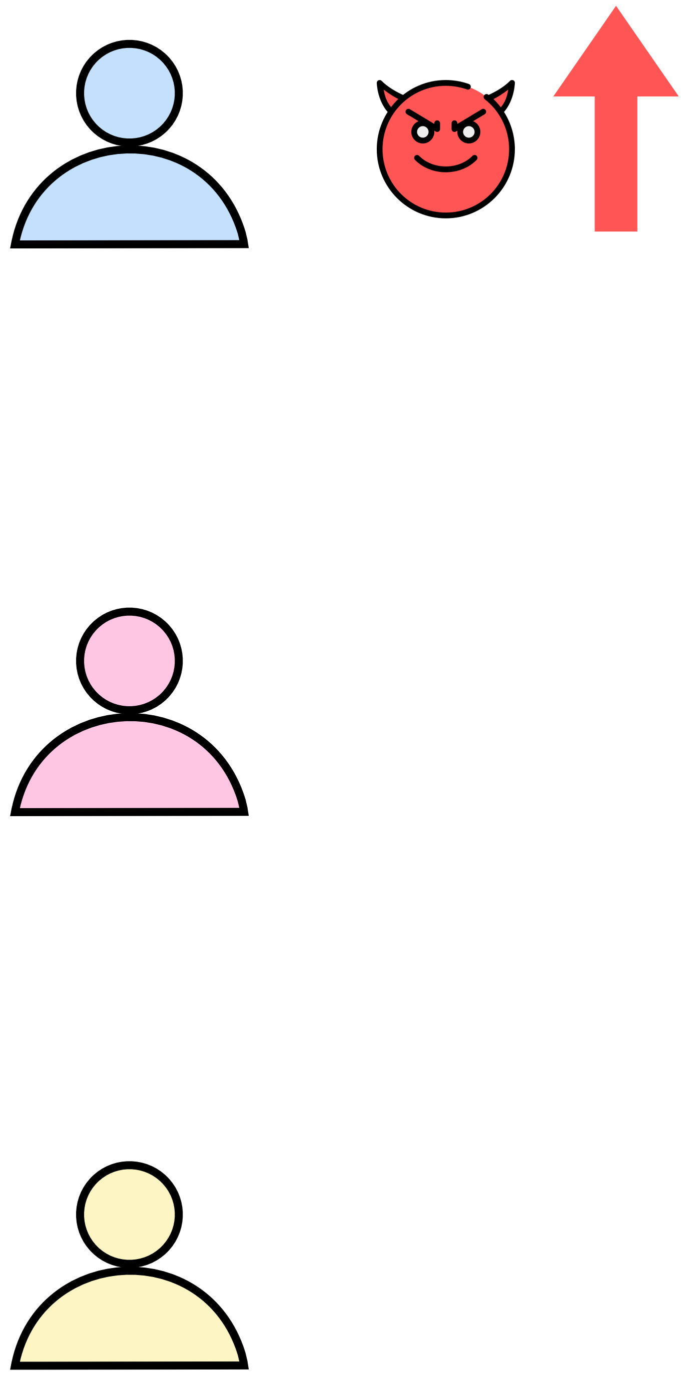
Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)



Karma provides powerful strategy-proofness properties

Strategy-proofness

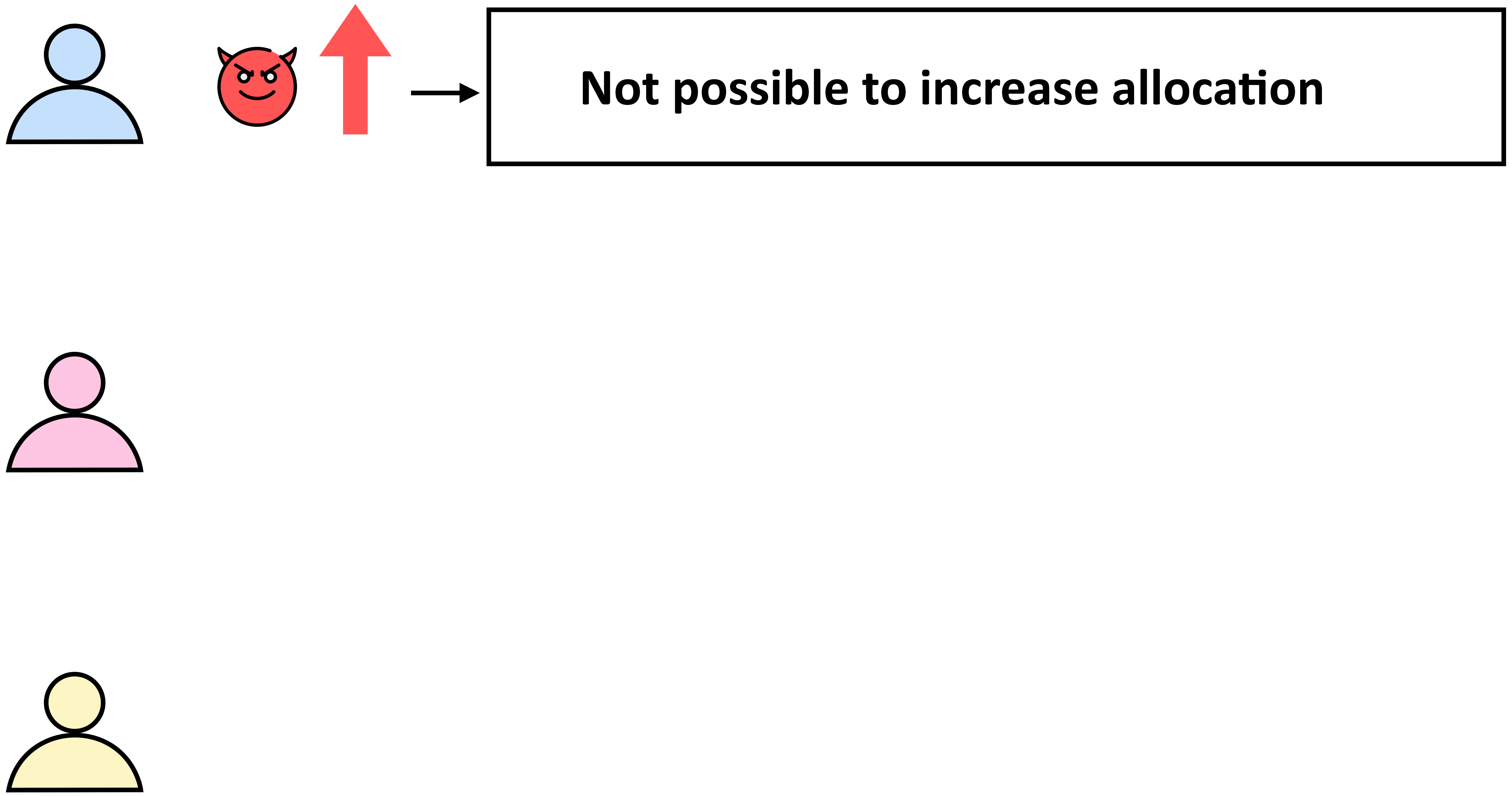
Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)



Karma provides powerful strategy-proofness properties

Strategy-proofness

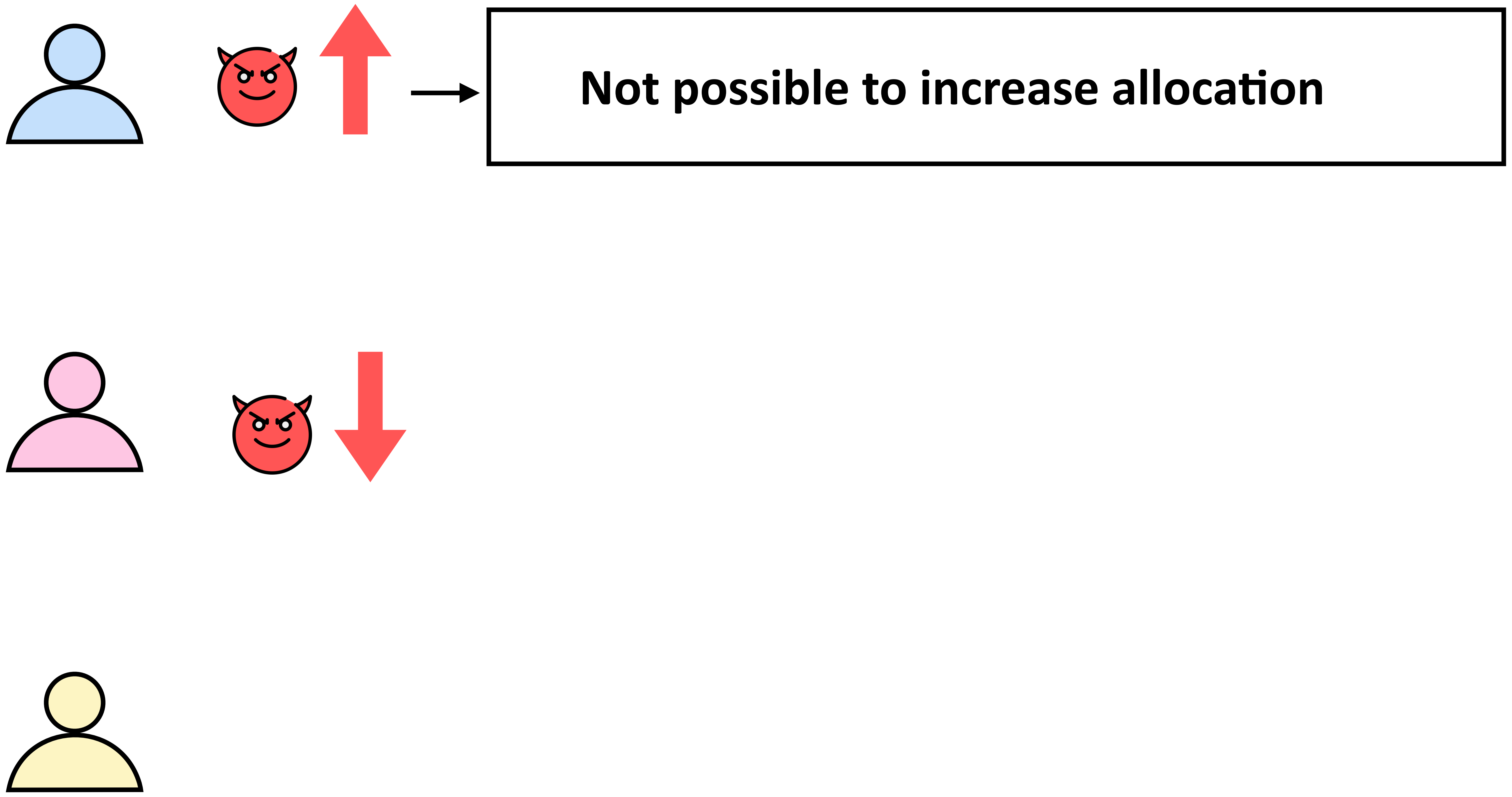
Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)



Karma provides powerful strategy-proofness properties

Strategy-proofness

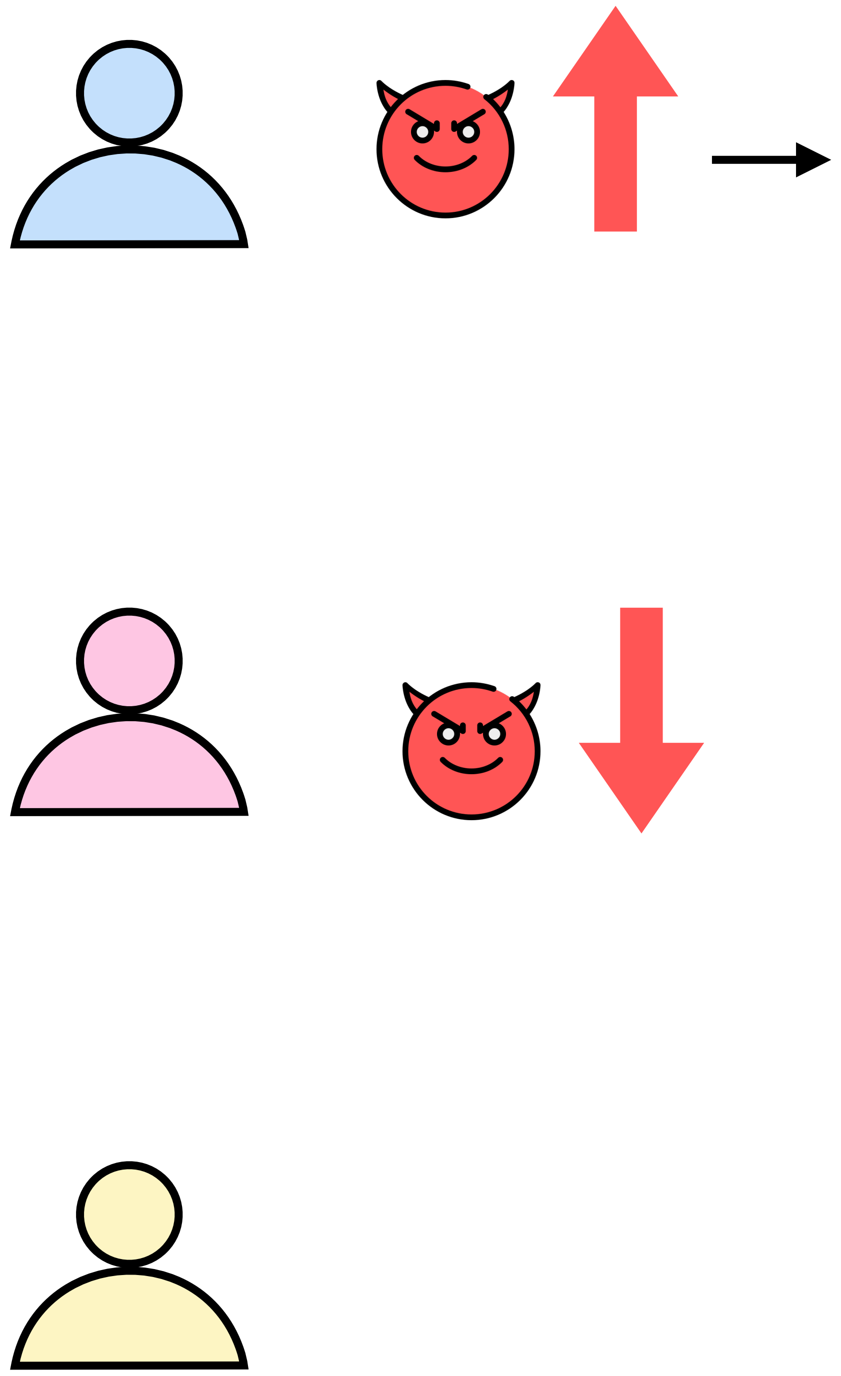
Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)



Karma provides powerful strategy-proofness properties

Strategy-proofness

Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)



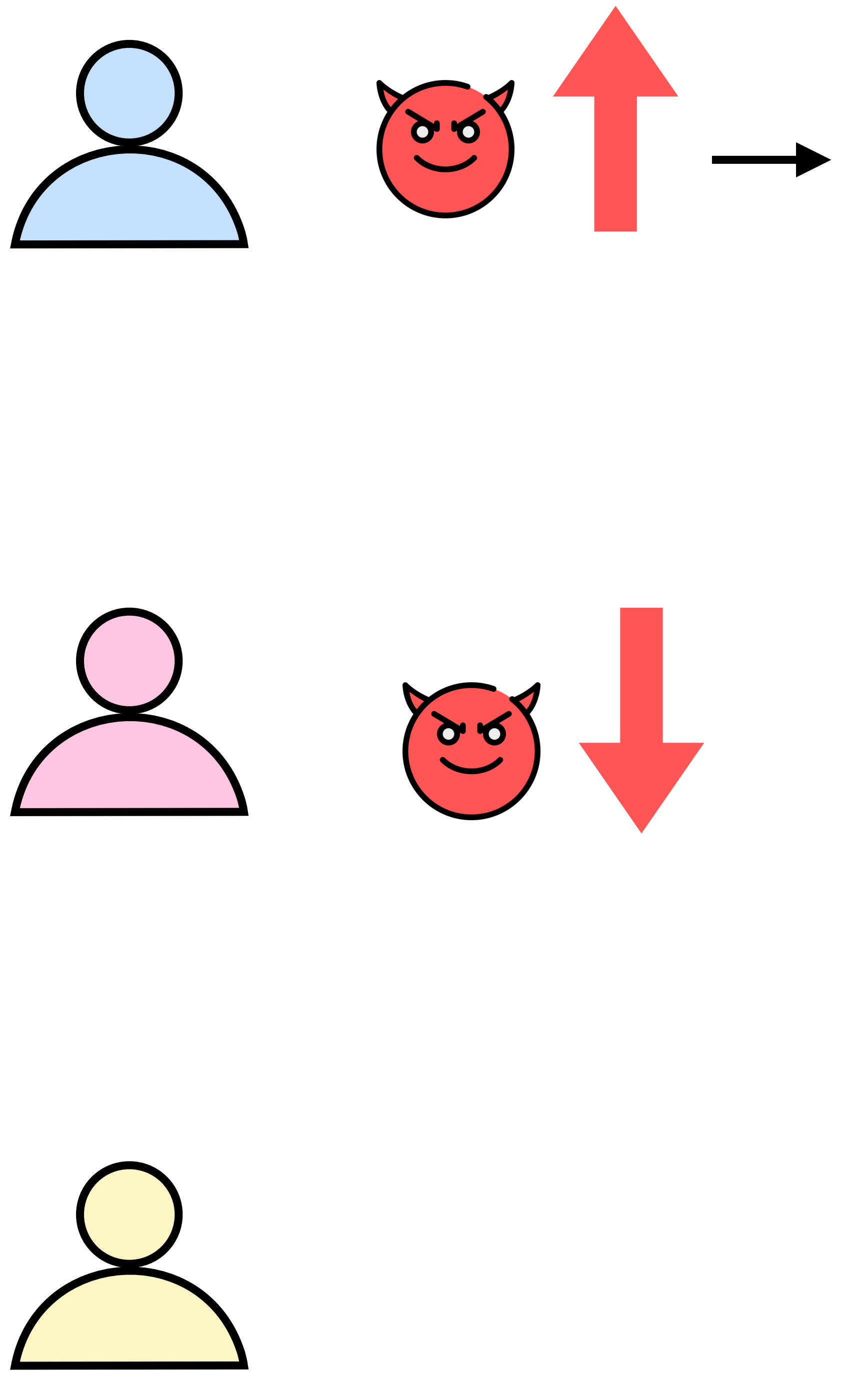
Not possible to increase allocation

Perfect knowledge of
future demands of **all users**

Karma provides powerful strategy-proofness properties

Strategy-proofness

Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)



Not possible to increase allocation

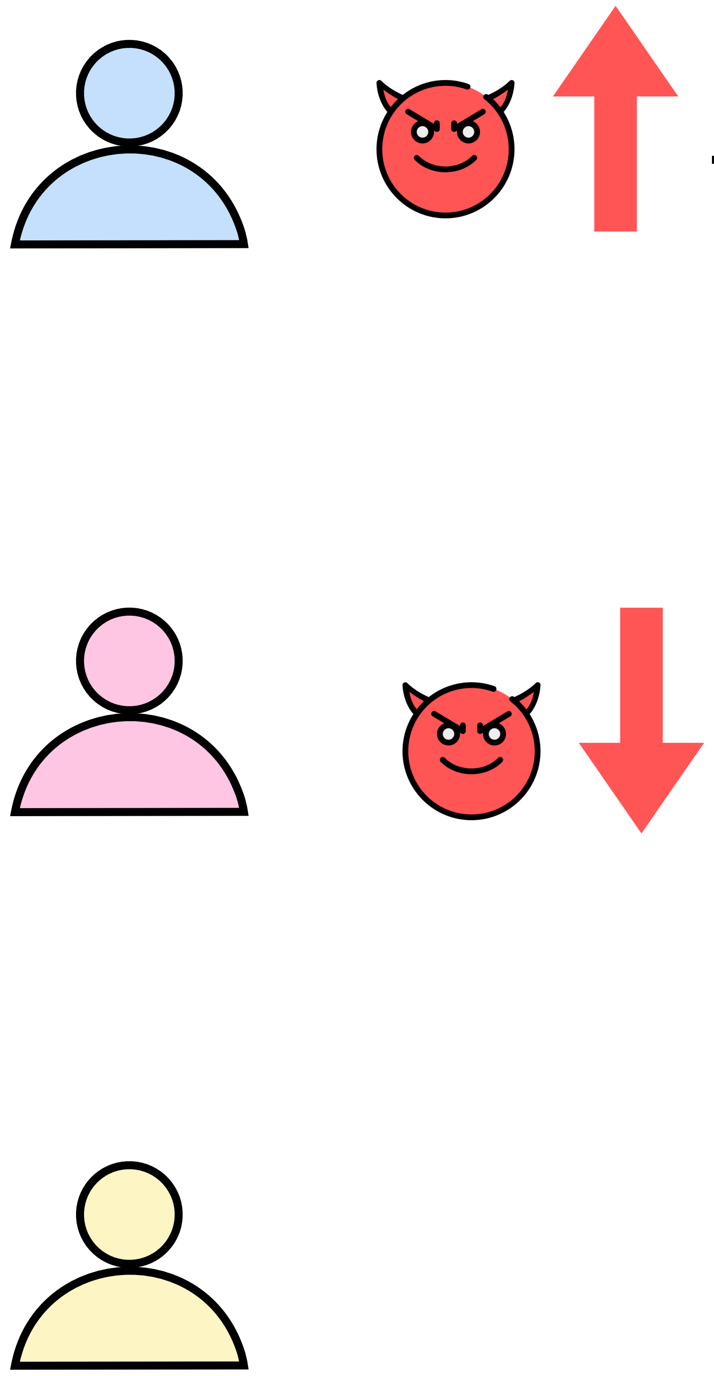
Perfect knowledge of
future demands of **all users**

No more than 1.5x increase in allocation

Karma provides powerful strategy-proofness properties

Strategy-proofness

Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)



Not possible to increase allocation

Perfect knowledge of
future demands of **all users**

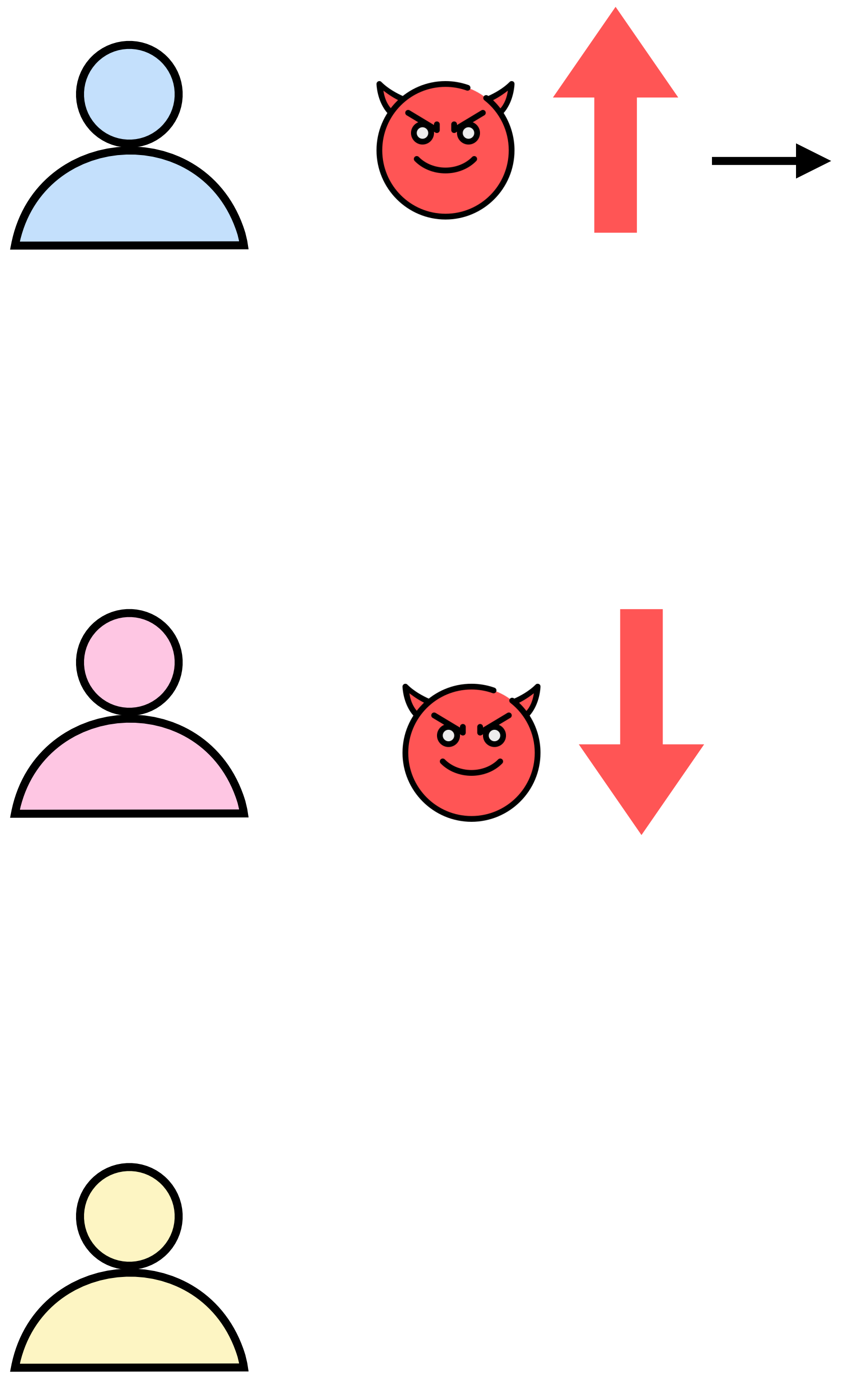
Far from realistic

No more than 1.5x increase in allocation

Karma provides powerful strategy-proofness properties

Strategy-proofness

Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)



Not possible to increase allocation

Perfect knowledge of
future demands of **all users**

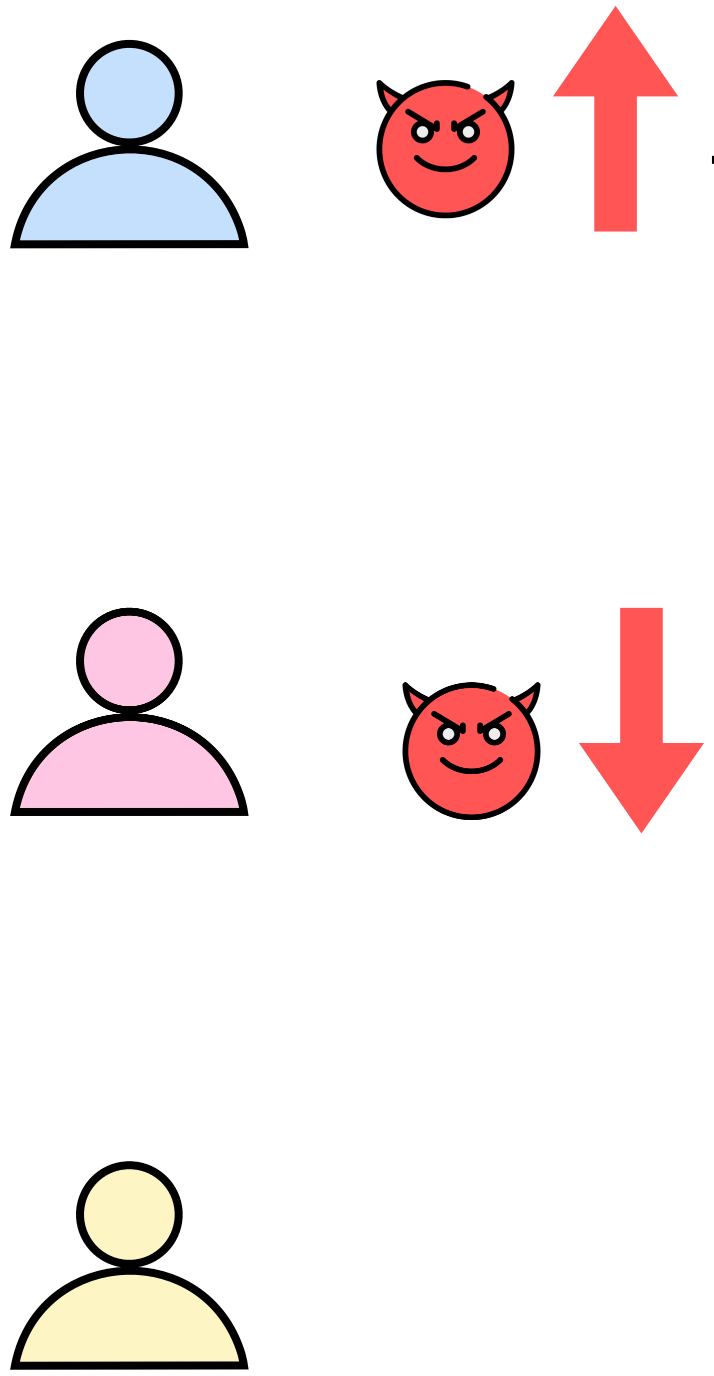
No more than 1.5x increase in allocation

Any imprecision in knowledge of
future demands of any user

Karma provides powerful strategy-proofness properties

Strategy-proofness

Selfish users cannot increase their allocation by lying
(selfish ≠ adversarial)



Not possible to increase allocation

Perfect knowledge of
future demands of **all users**

No more than 1.5x increase in allocation

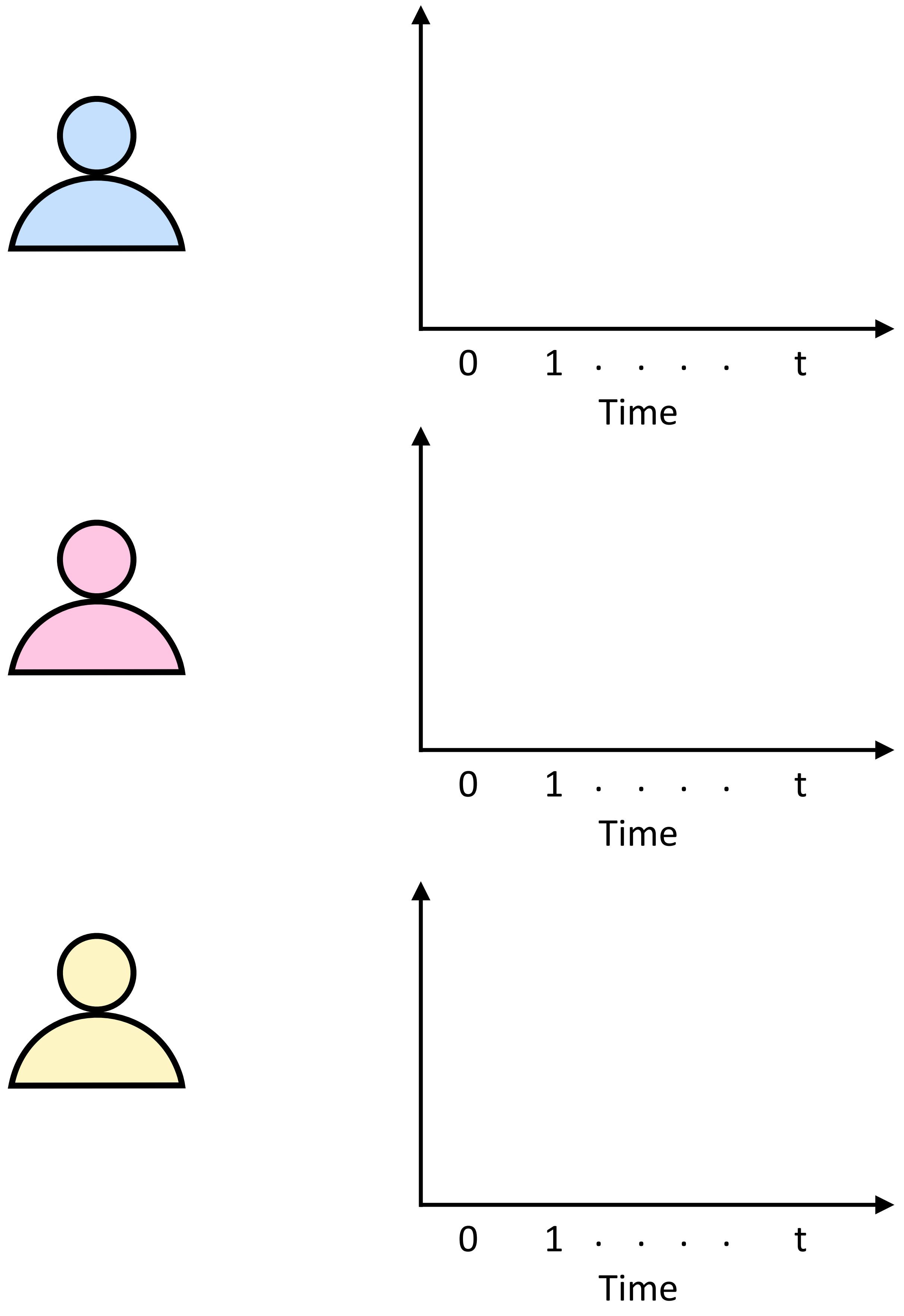
<u>Any</u> imprecision in knowledge of
future demands of any user

As much as Ω(n) factor decrease in allocation

Karma maximizes the minimum total allocation (given past allocations)

Fairness

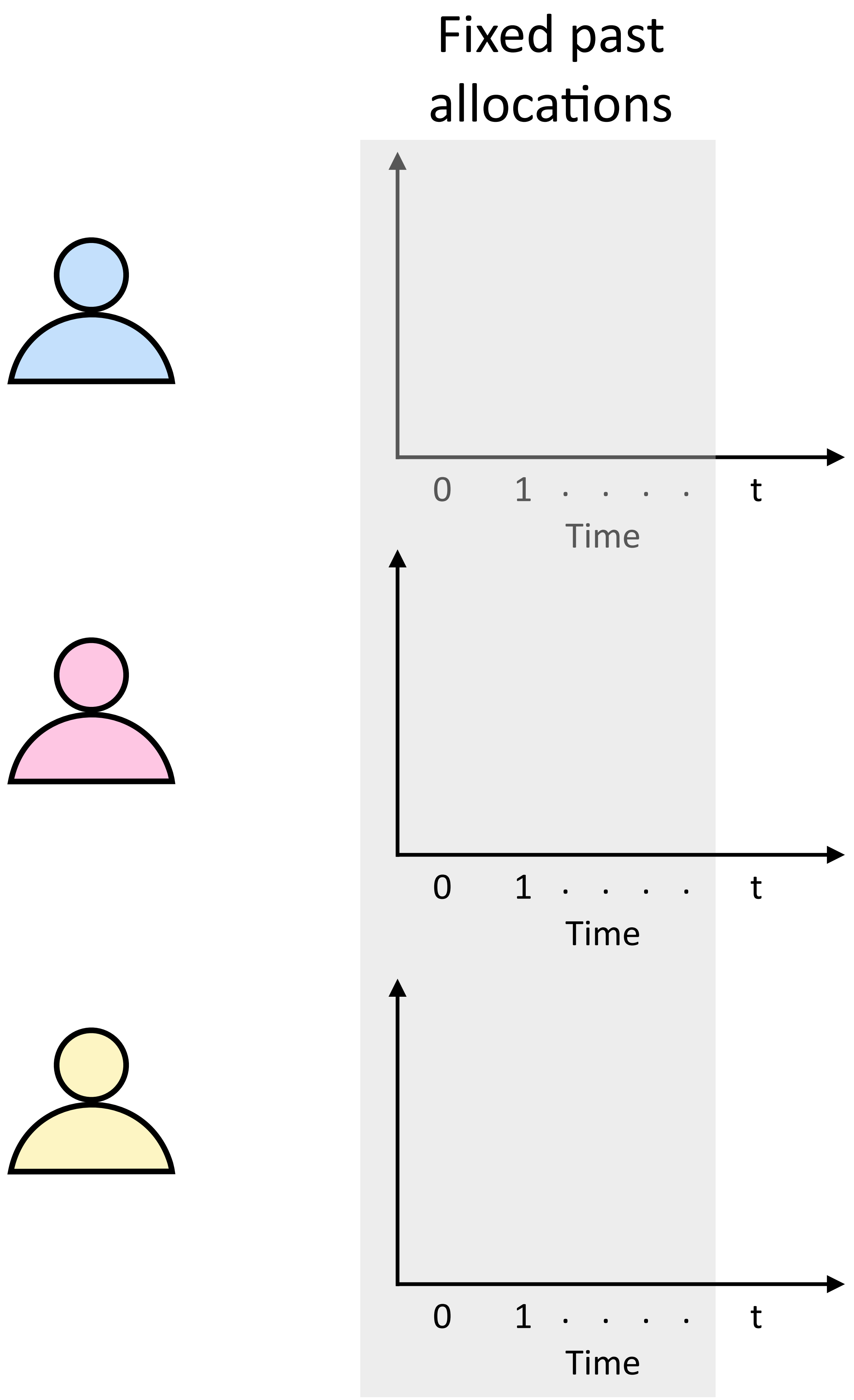
Balanced resource allocations across users



Karma maximizes the minimum total allocation (given past allocations)

Fairness

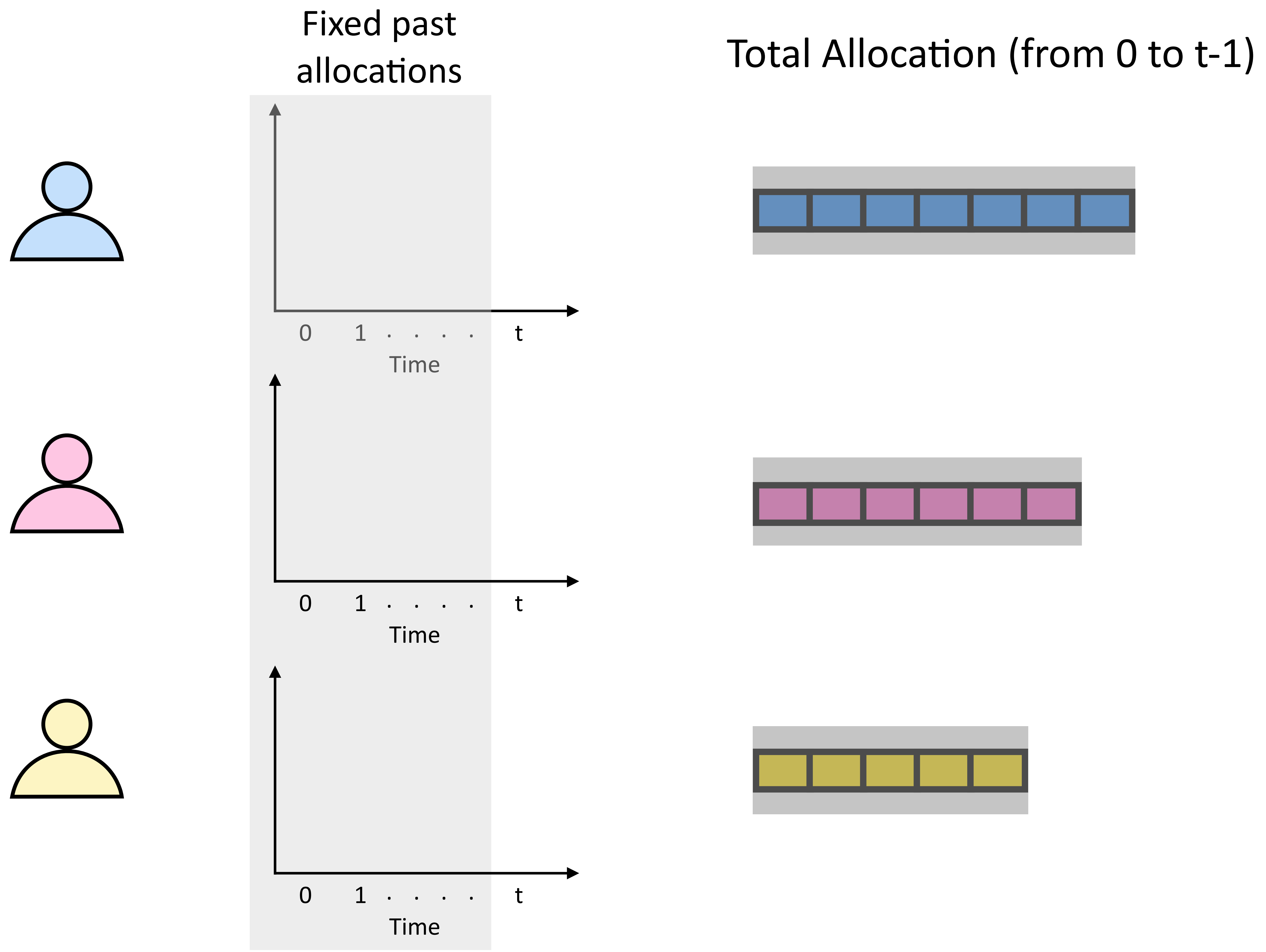
Balanced resource allocations across users



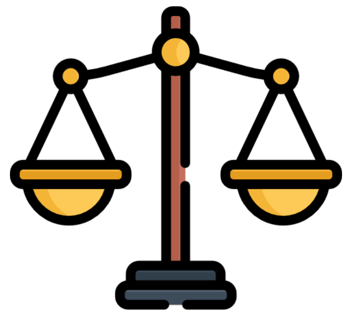
Karma maximizes the minimum total allocation (given past allocations)

Fairness

Balanced resource allocations across users

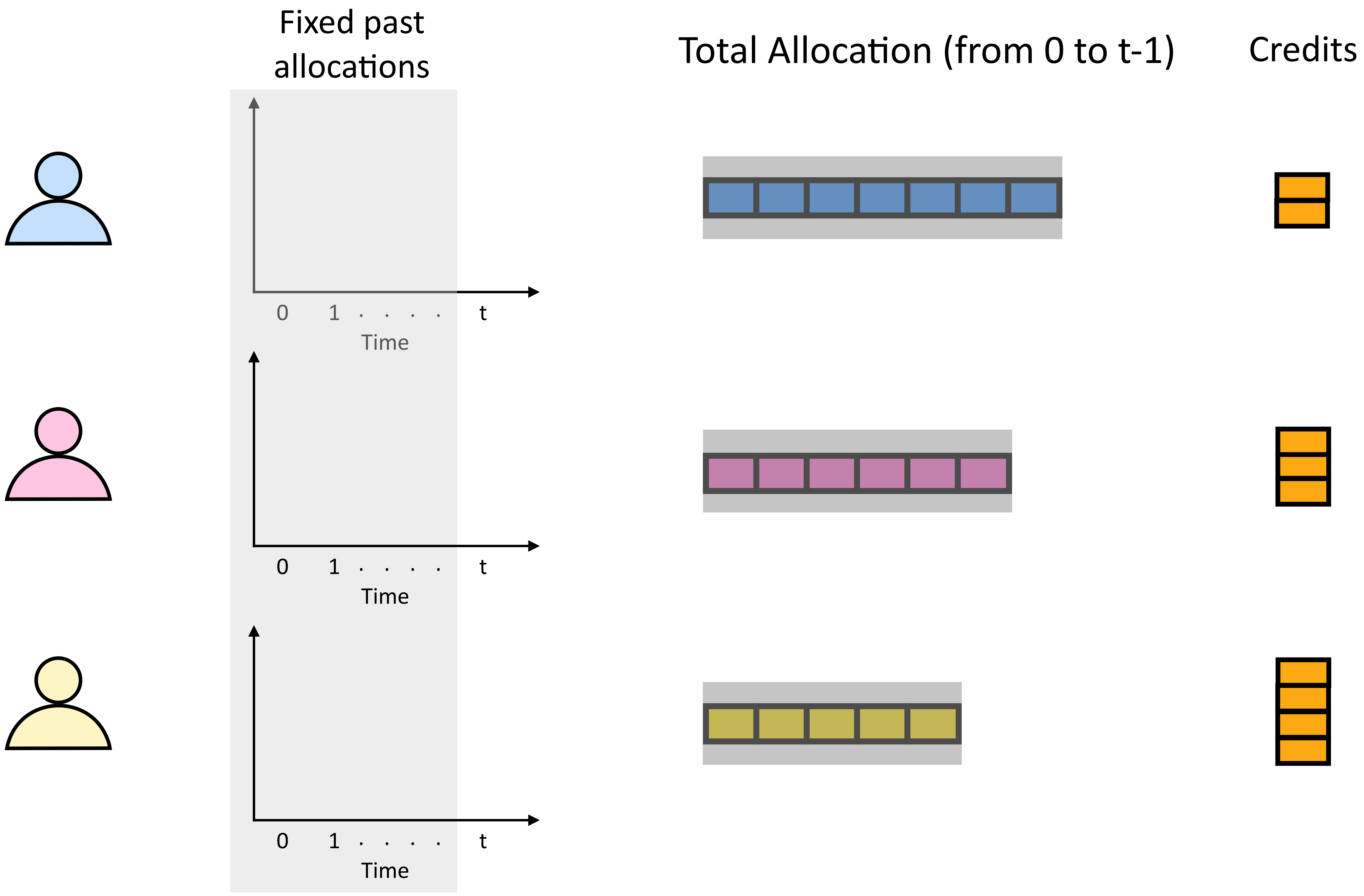


Karma maximizes the minimum total allocation (given past allocations)



Fairness

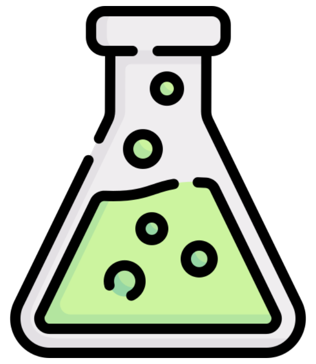
Balanced resource allocations across users



Karma: Resource Allocation for Dynamic Demands

New credit-based resource allocation algorithm

Strong theoretical guarantees



Prototype implementation and evaluation

Karma implementation & evaluation

Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys'22])

Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys'22])

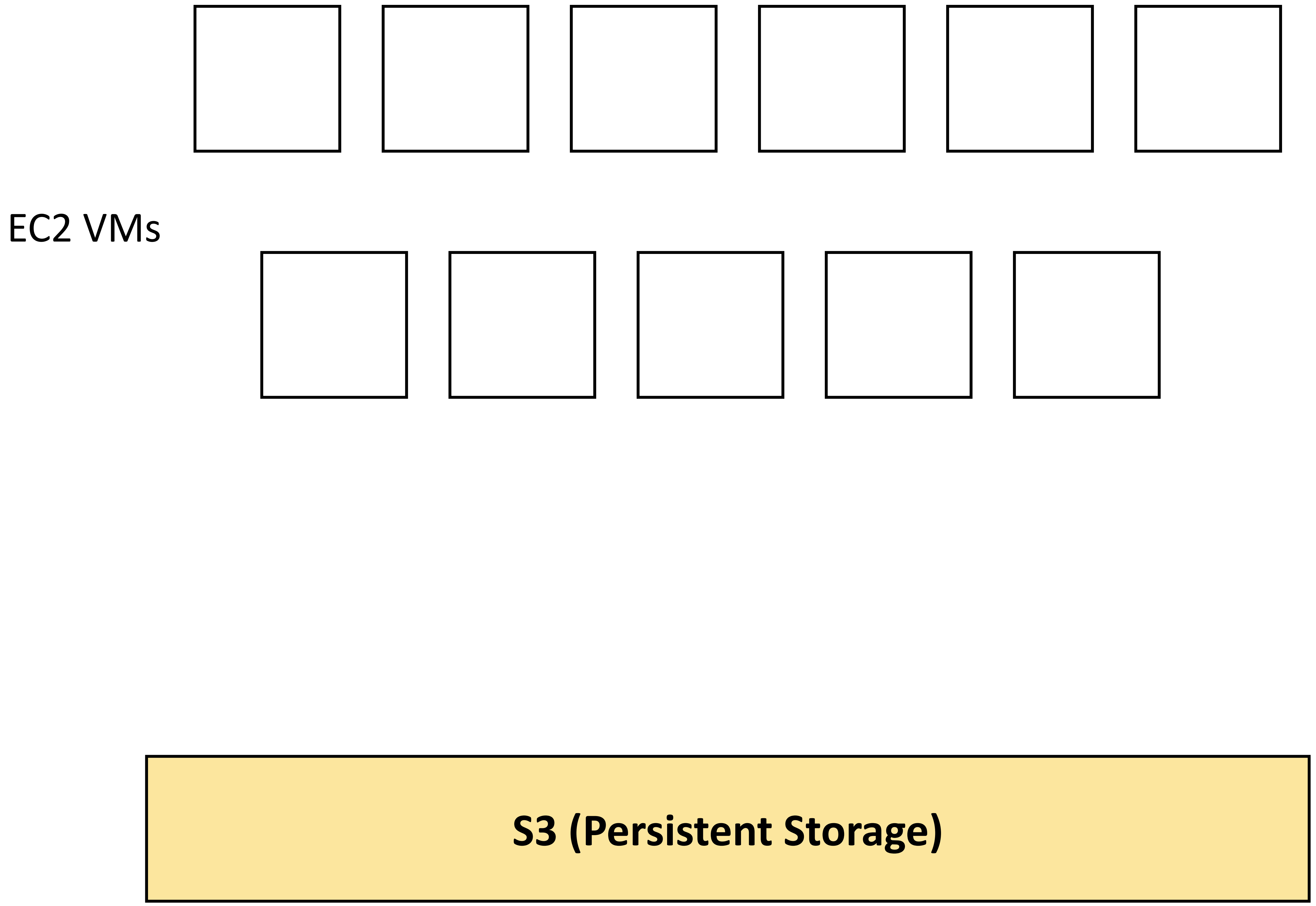
Evaluated in-memory key-value cache scenario

Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys'22])

Evaluated in-memory key-value cache scenario

Experimental Setup

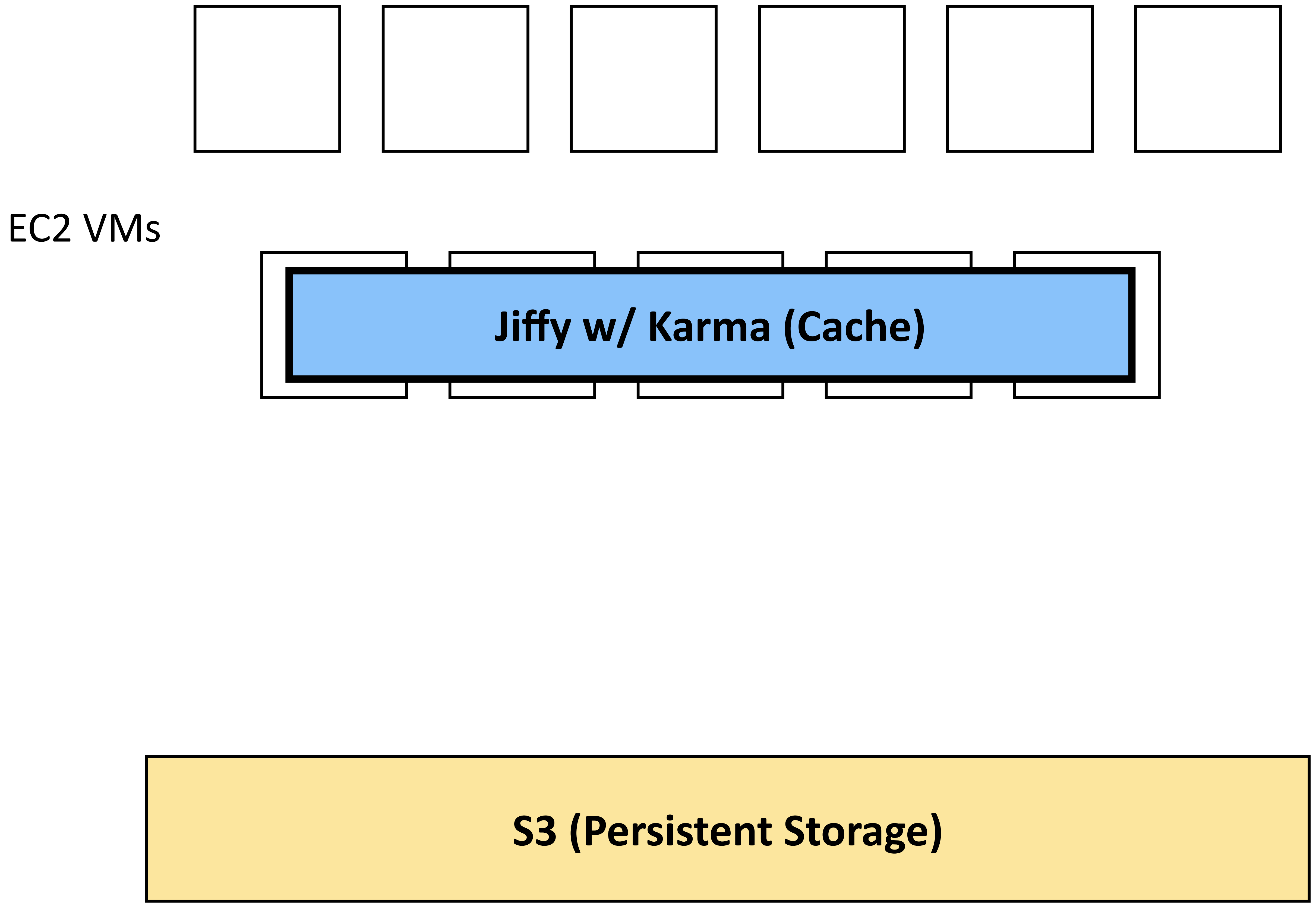


Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys'22])

Evaluated in-memory key-value cache scenario

Experimental Setup

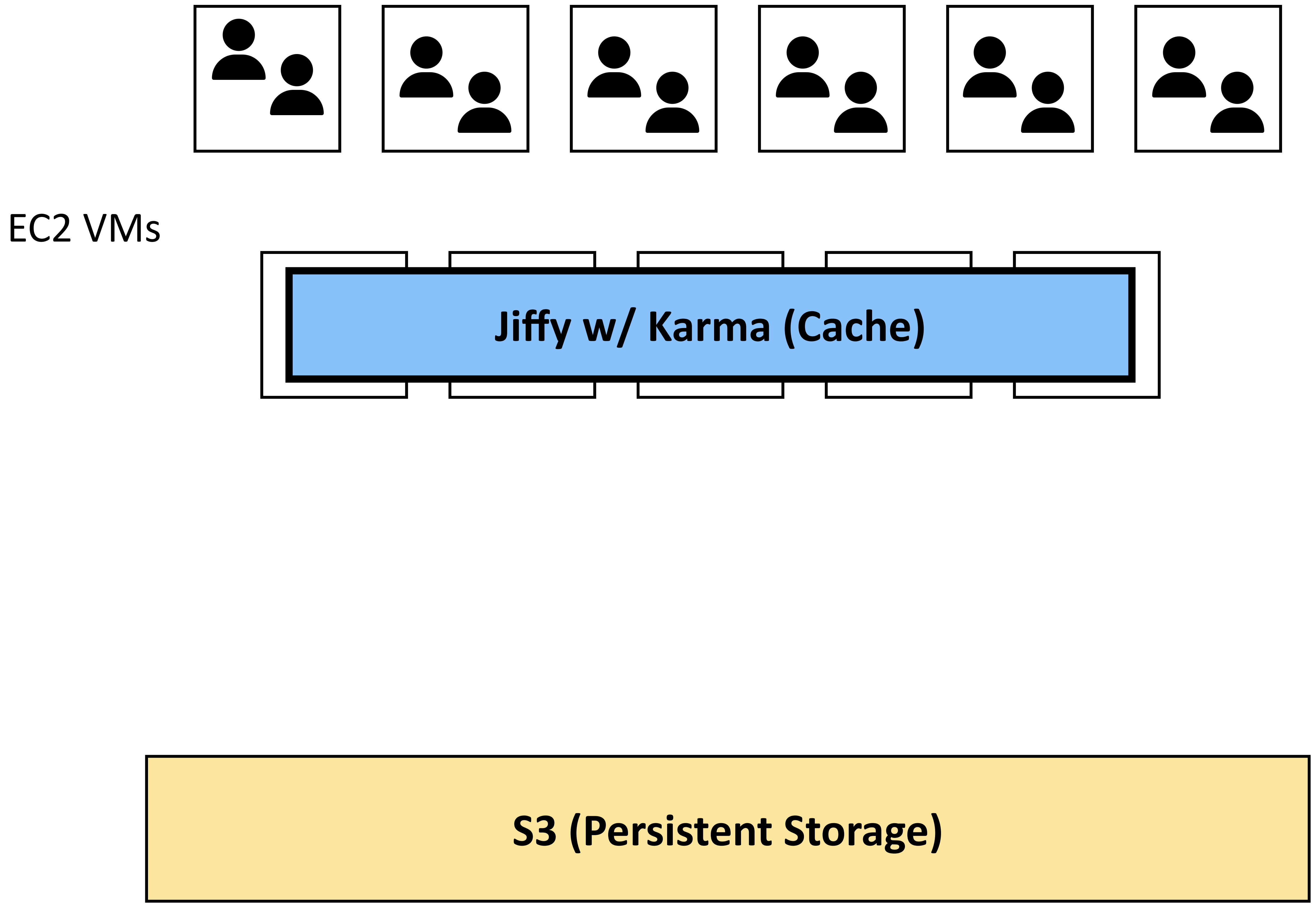


Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys'22])

Evaluated in-memory key-value cache scenario

Experimental Setup

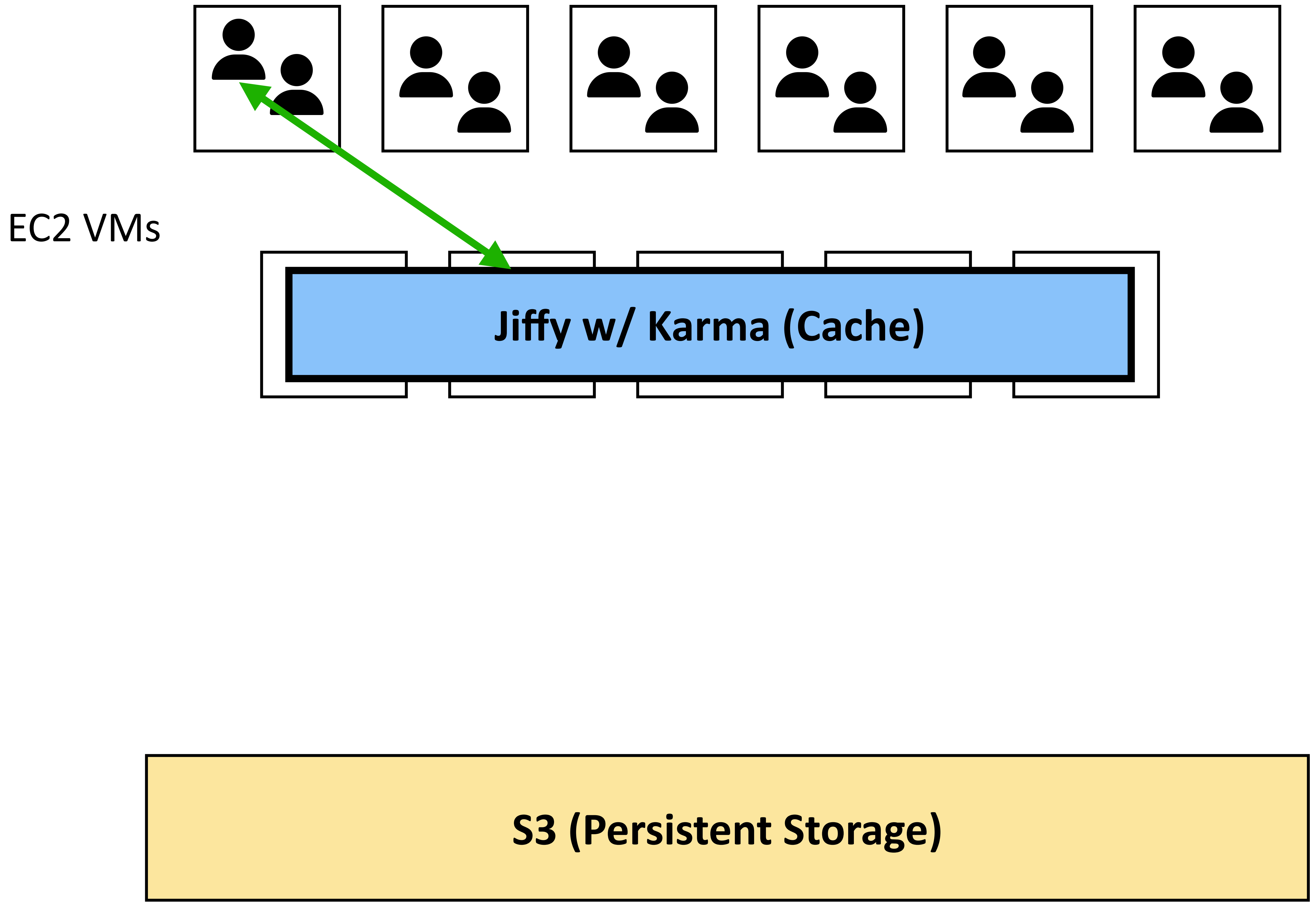


Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys'22])

Evaluated in-memory key-value cache scenario

Experimental Setup

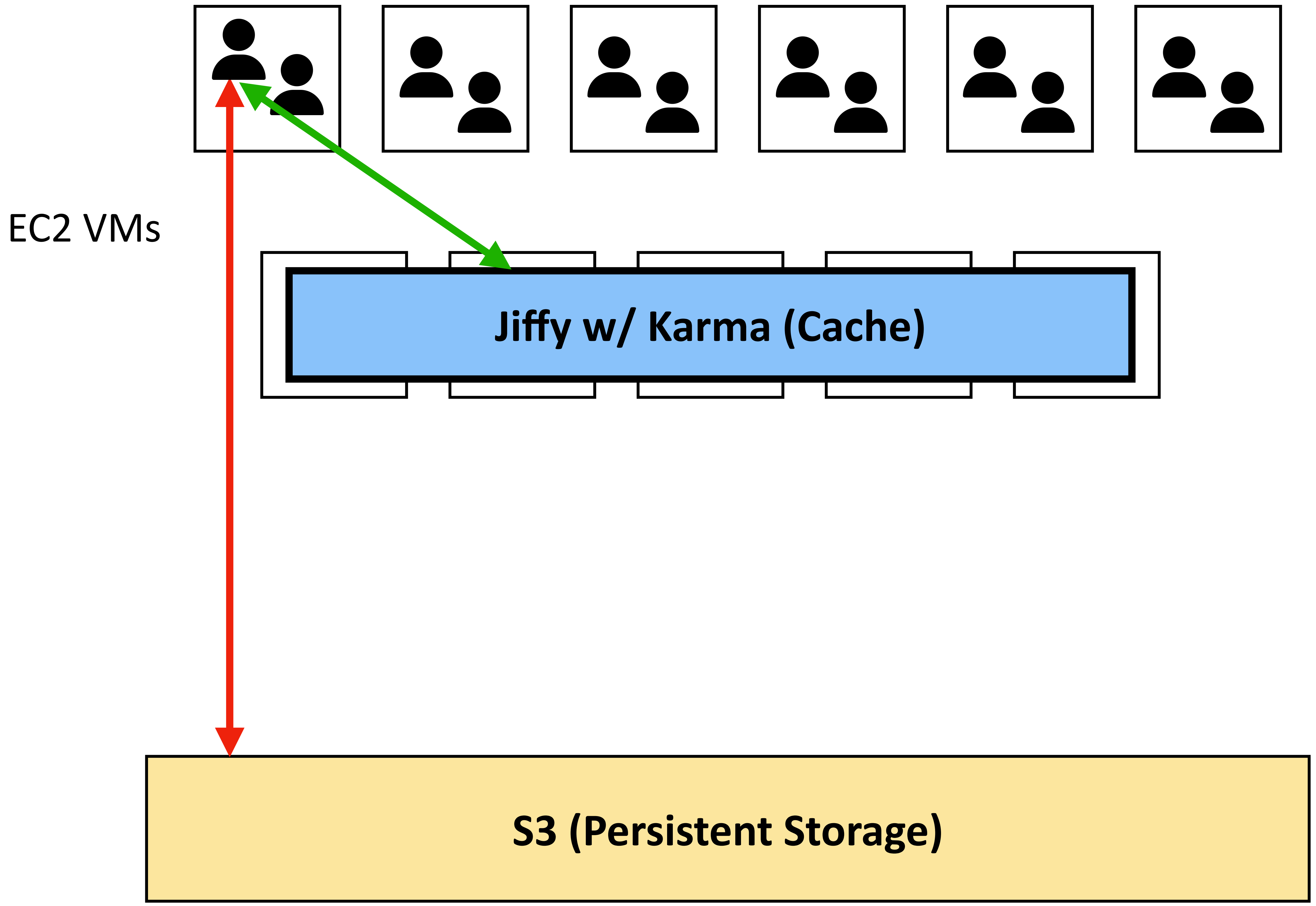


Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys'22])

Evaluated in-memory key-value cache scenario

Experimental Setup

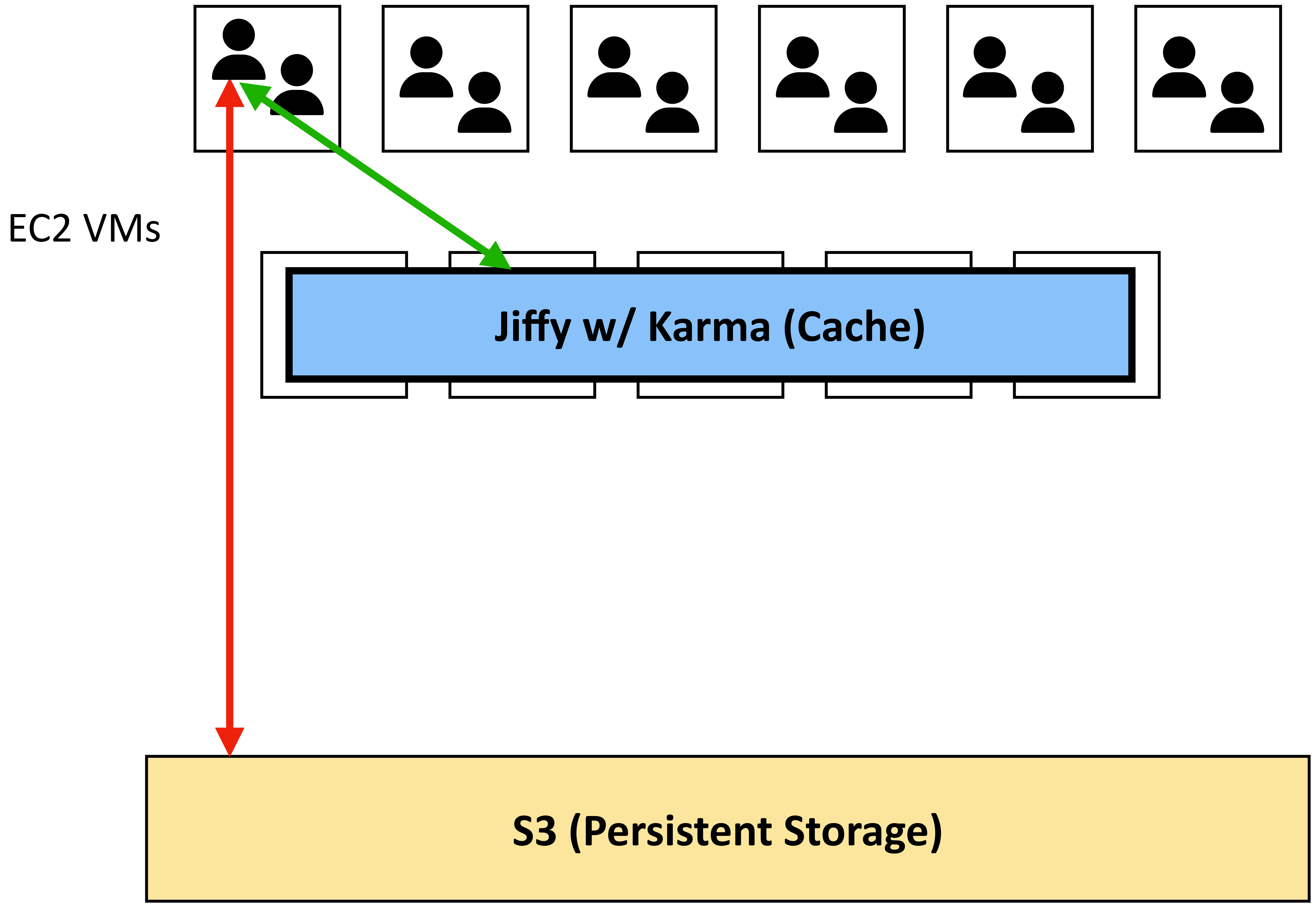


Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys'22])

Evaluated in-memory key-value cache scenario

Experimental Setup



Workload

Dynamic demands -> vary working set size

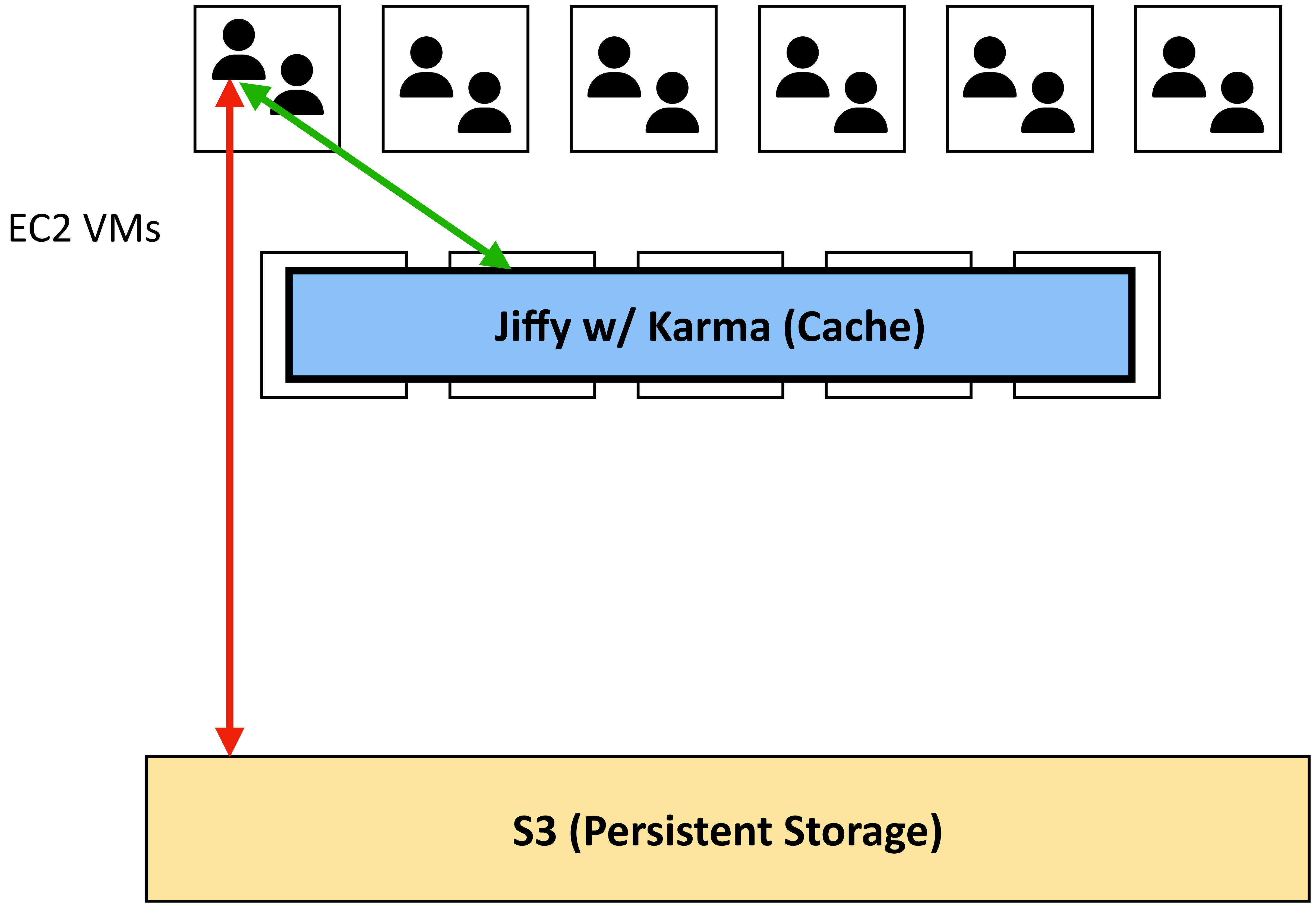
Demands taken from Snowflake dataset

Karma implementation & evaluation

Implemented in distributed elastic memory system (Jiffy [Eurosys'22])

Evaluated in-memory key-value cache scenario

Experimental Setup



Workload

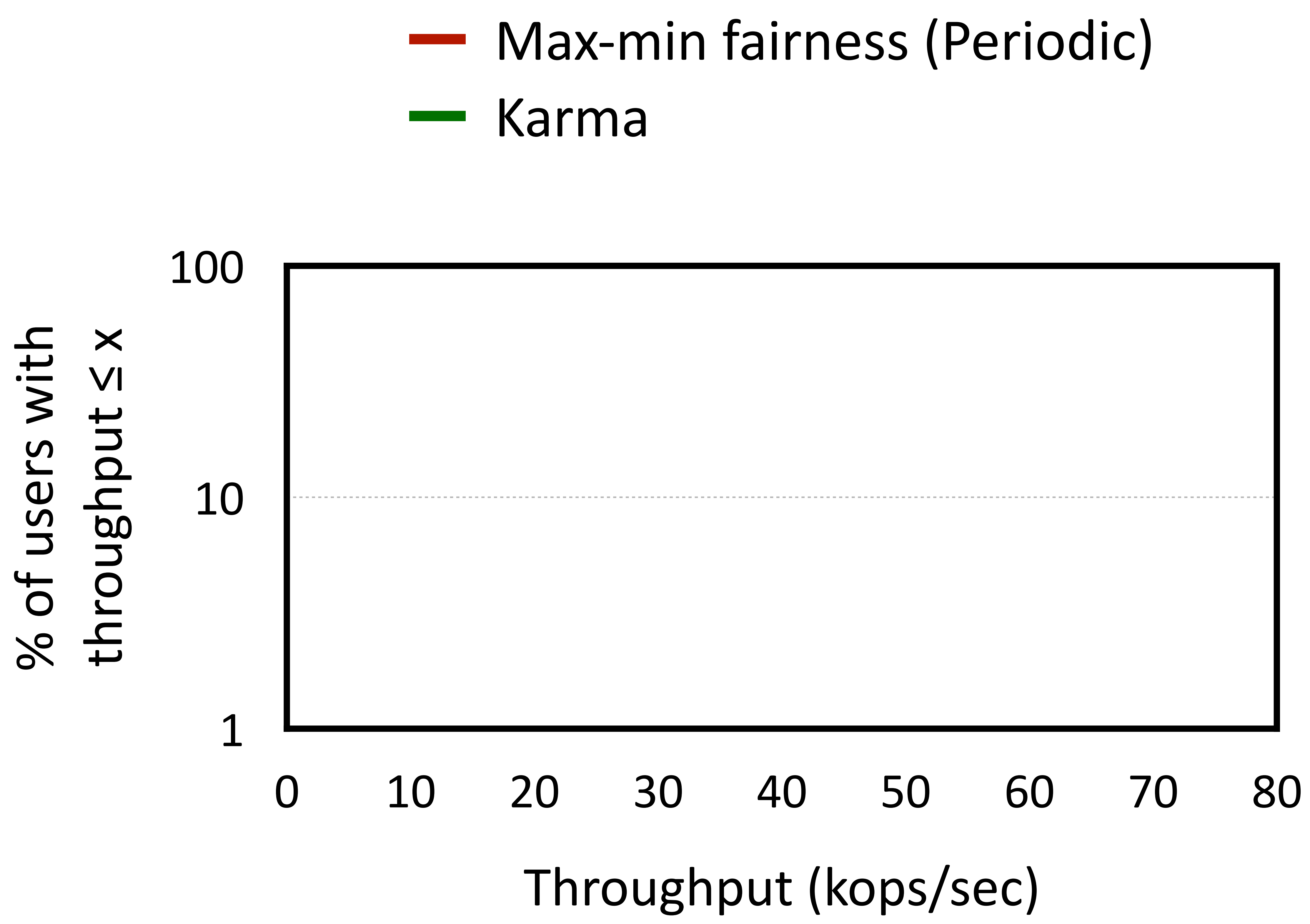
Dynamic demands -> vary working set size

Demands taken from Snowflake dataset

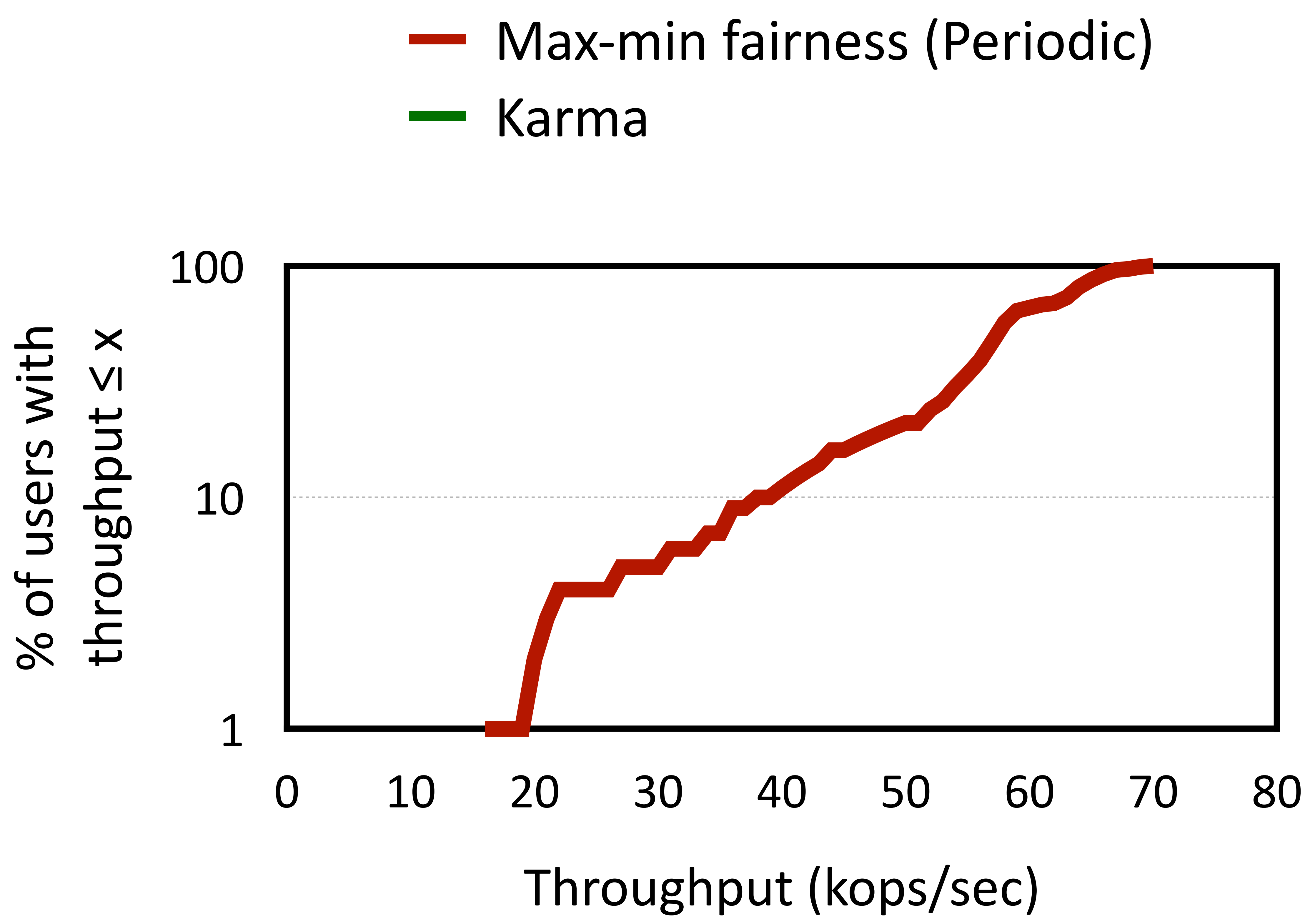
(See paper for details)

Glimpse of Karma evaluation results

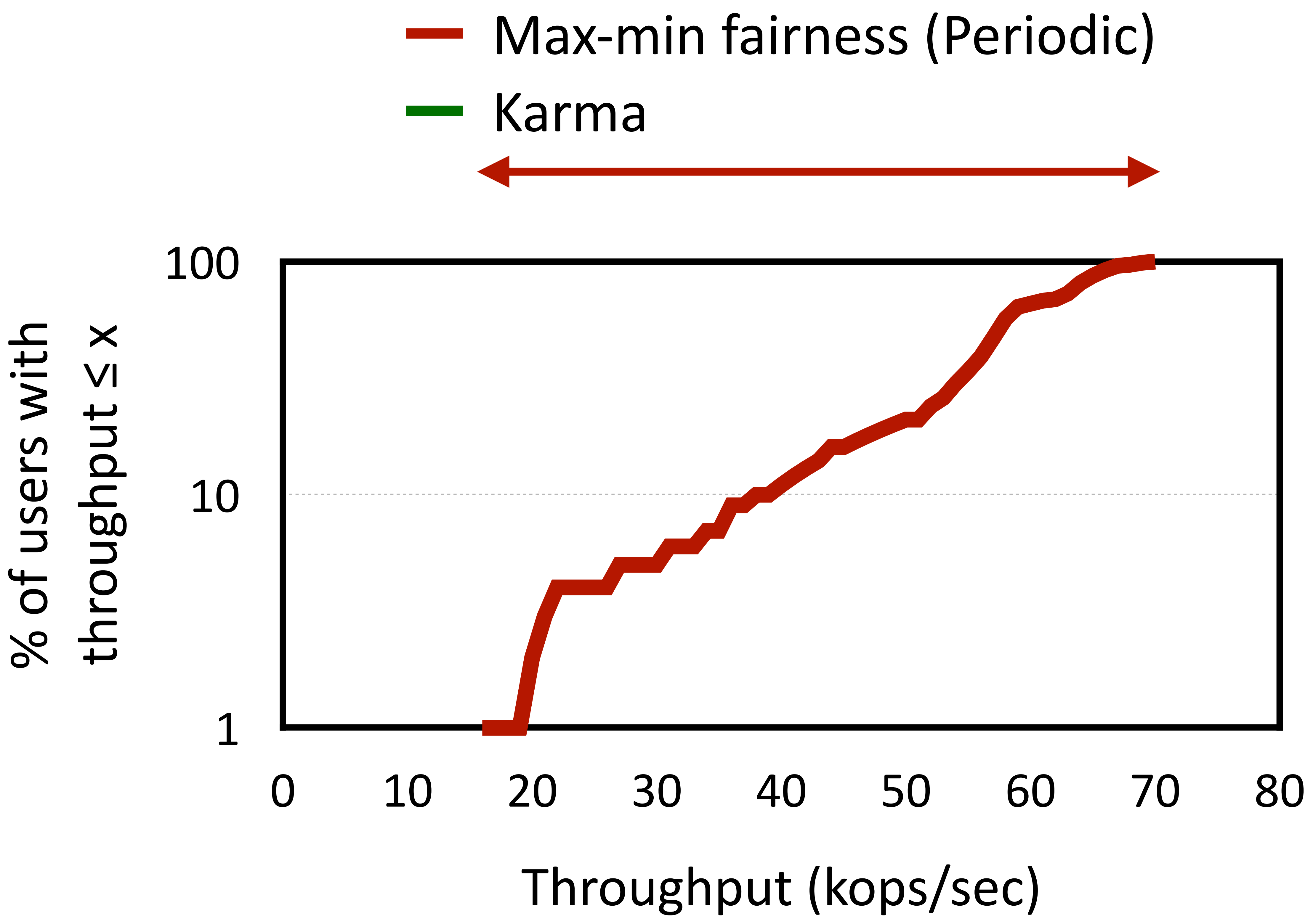
Glimpse of Karma evaluation results



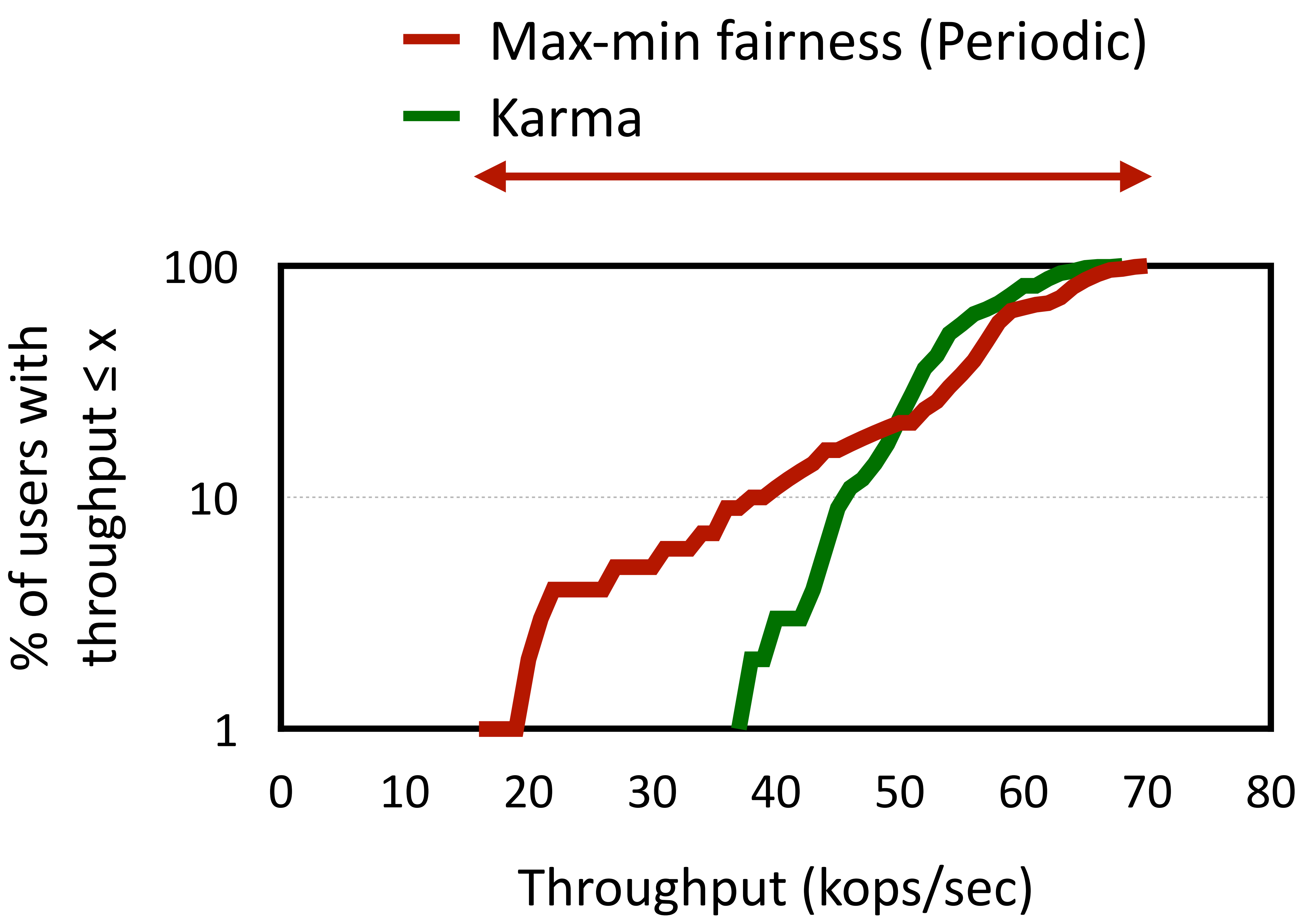
Glimpse of Karma evaluation results



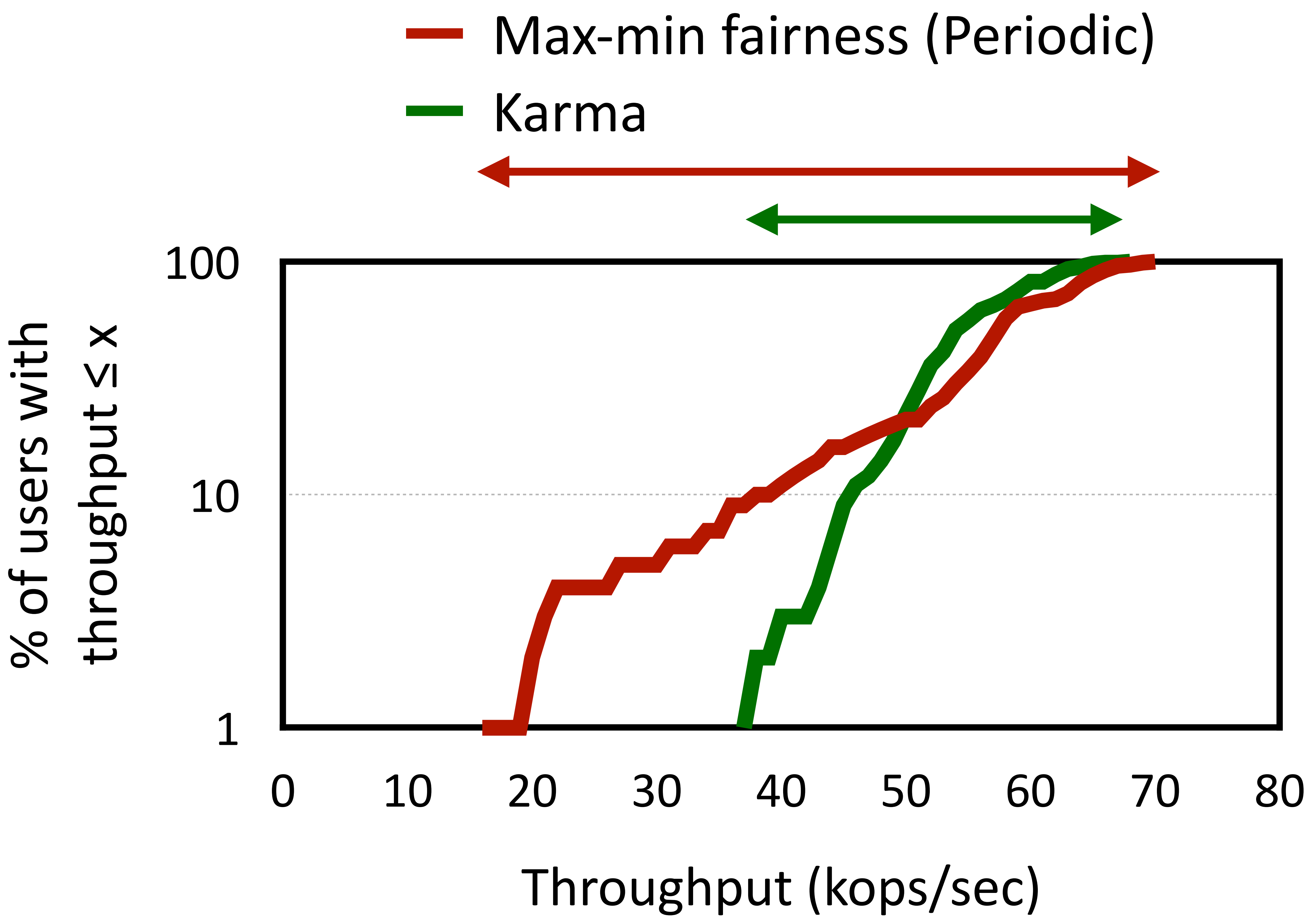
Glimpse of Karma evaluation results



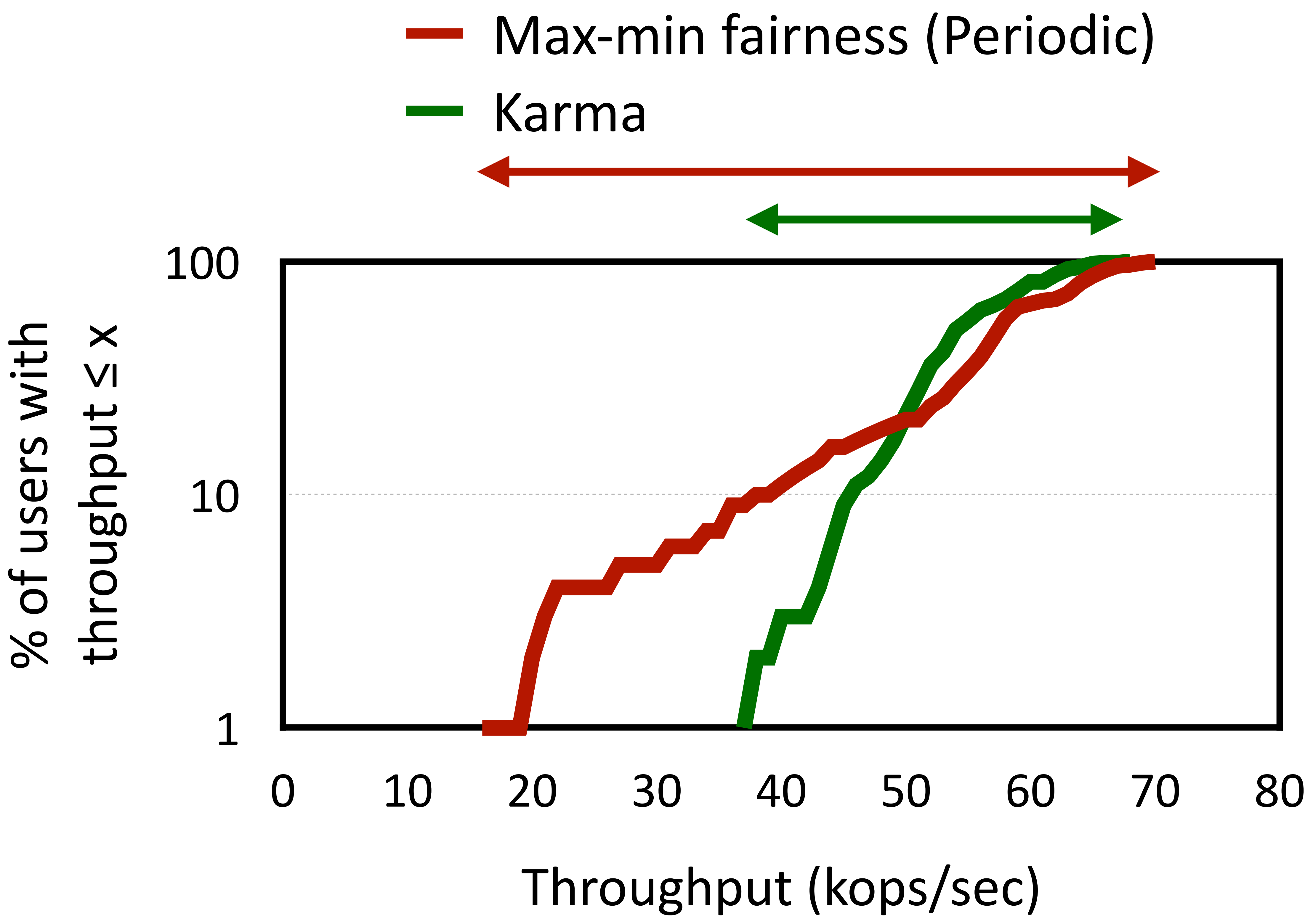
Glimpse of Karma evaluation results

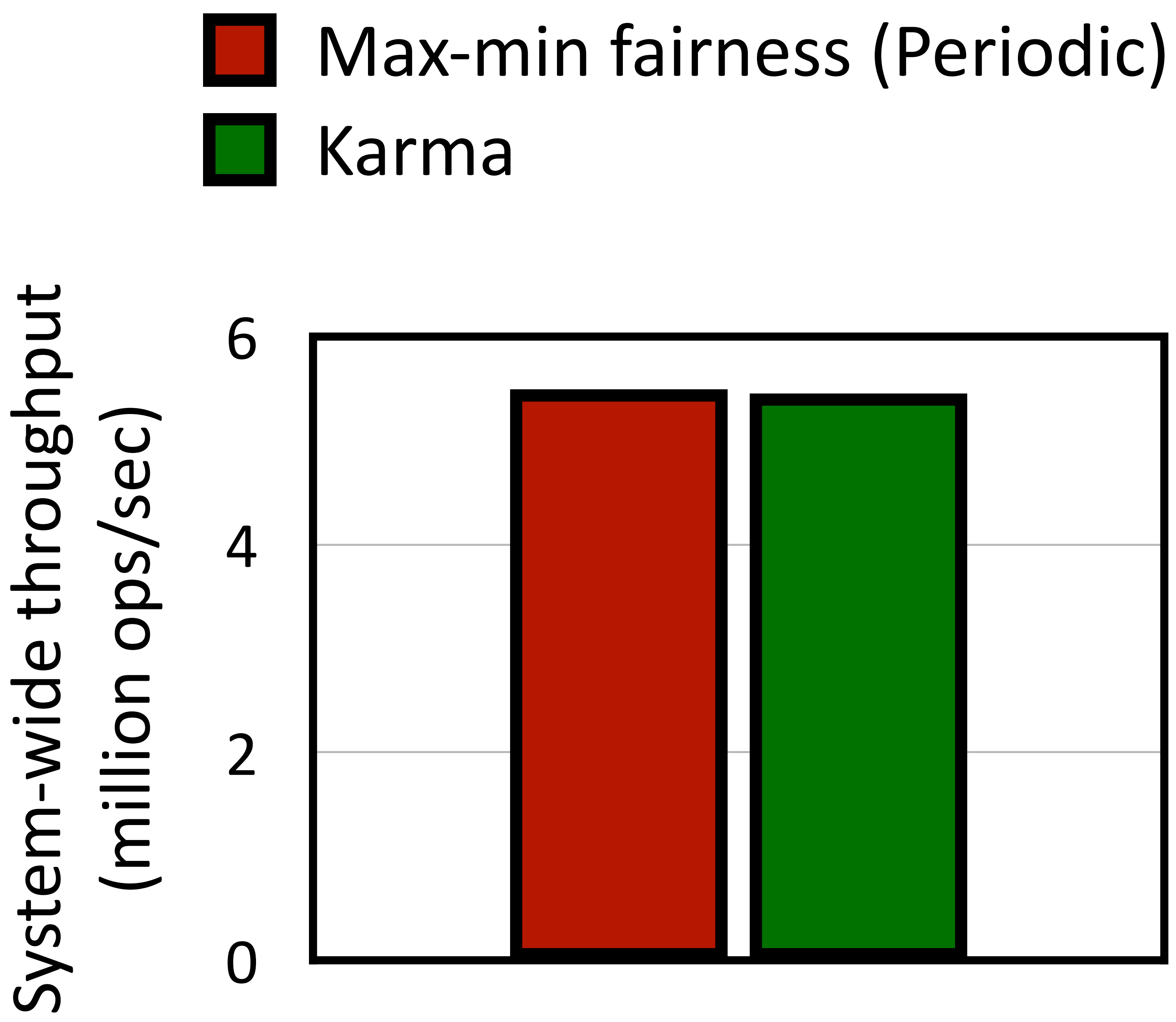


Glimpse of Karma evaluation results

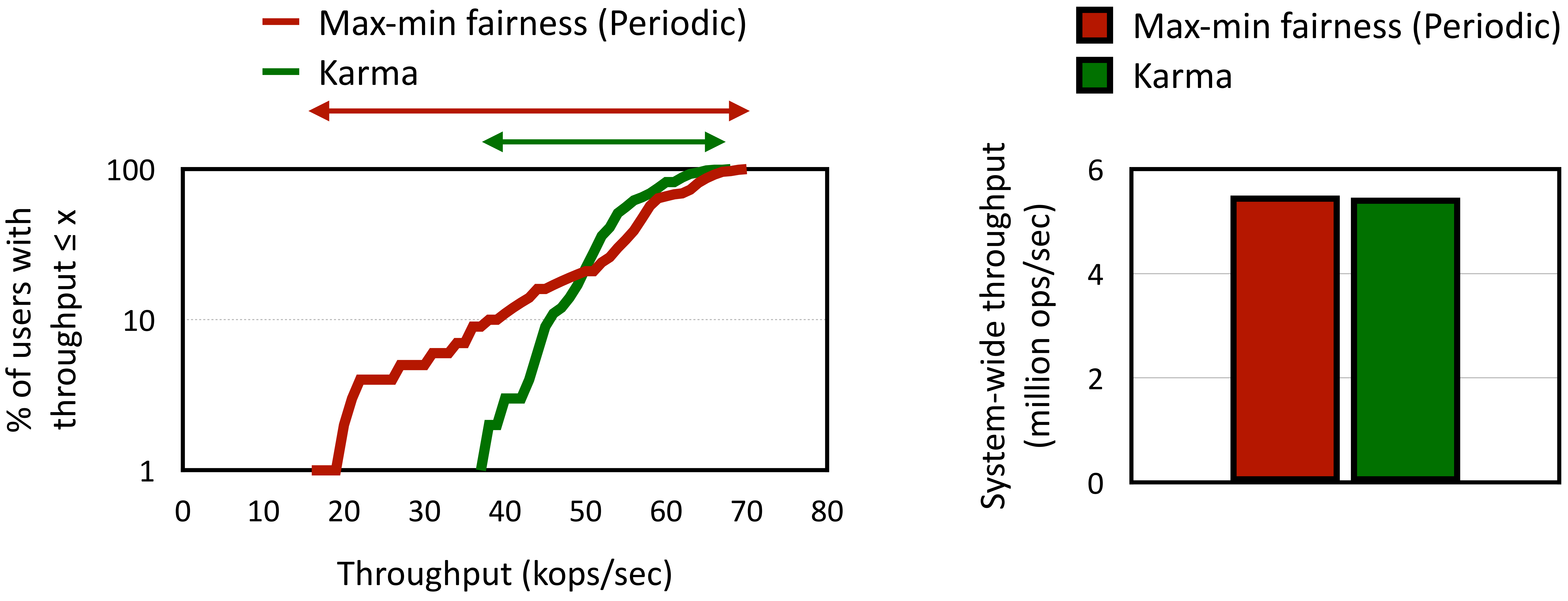


Glimpse of Karma evaluation results



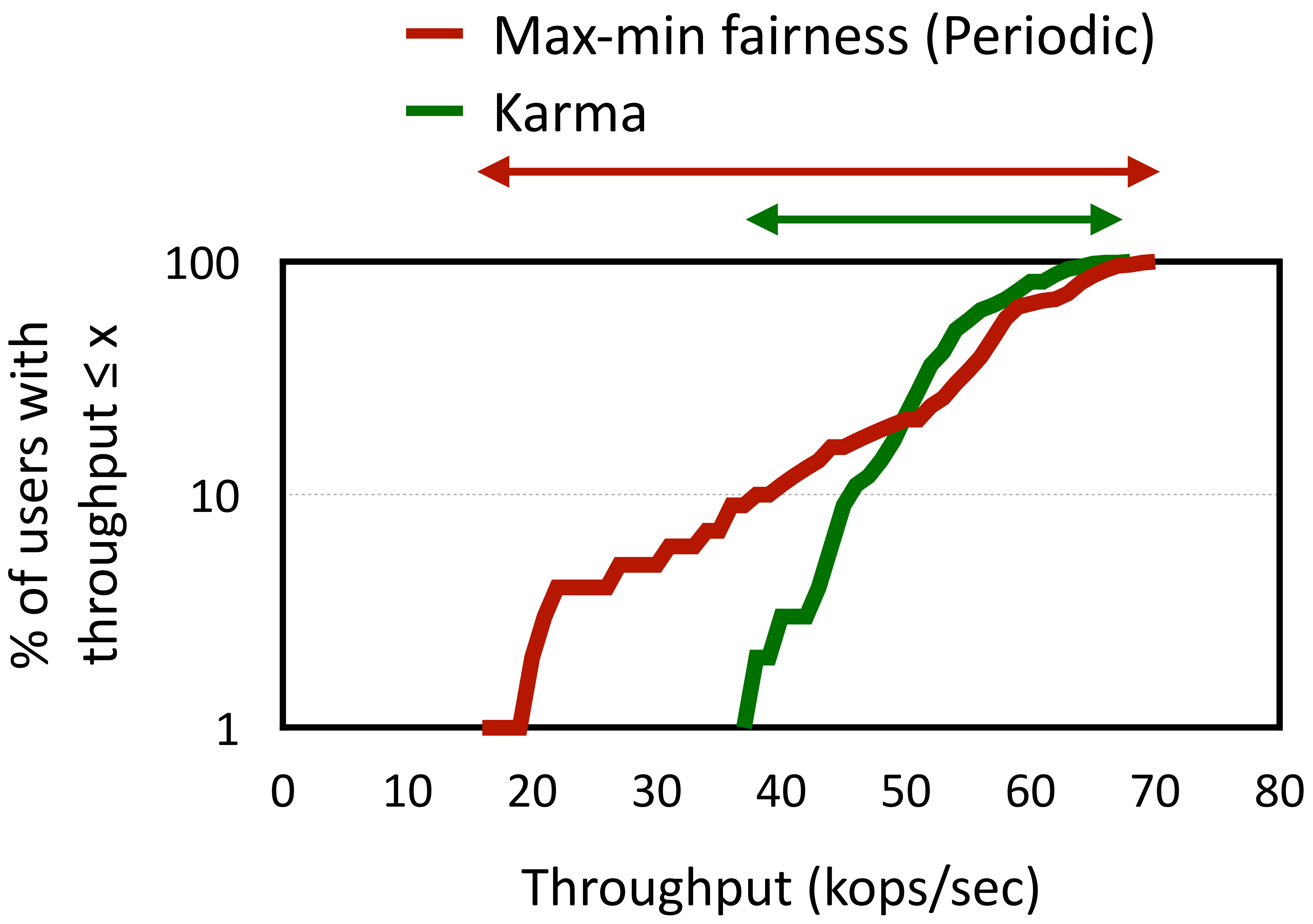


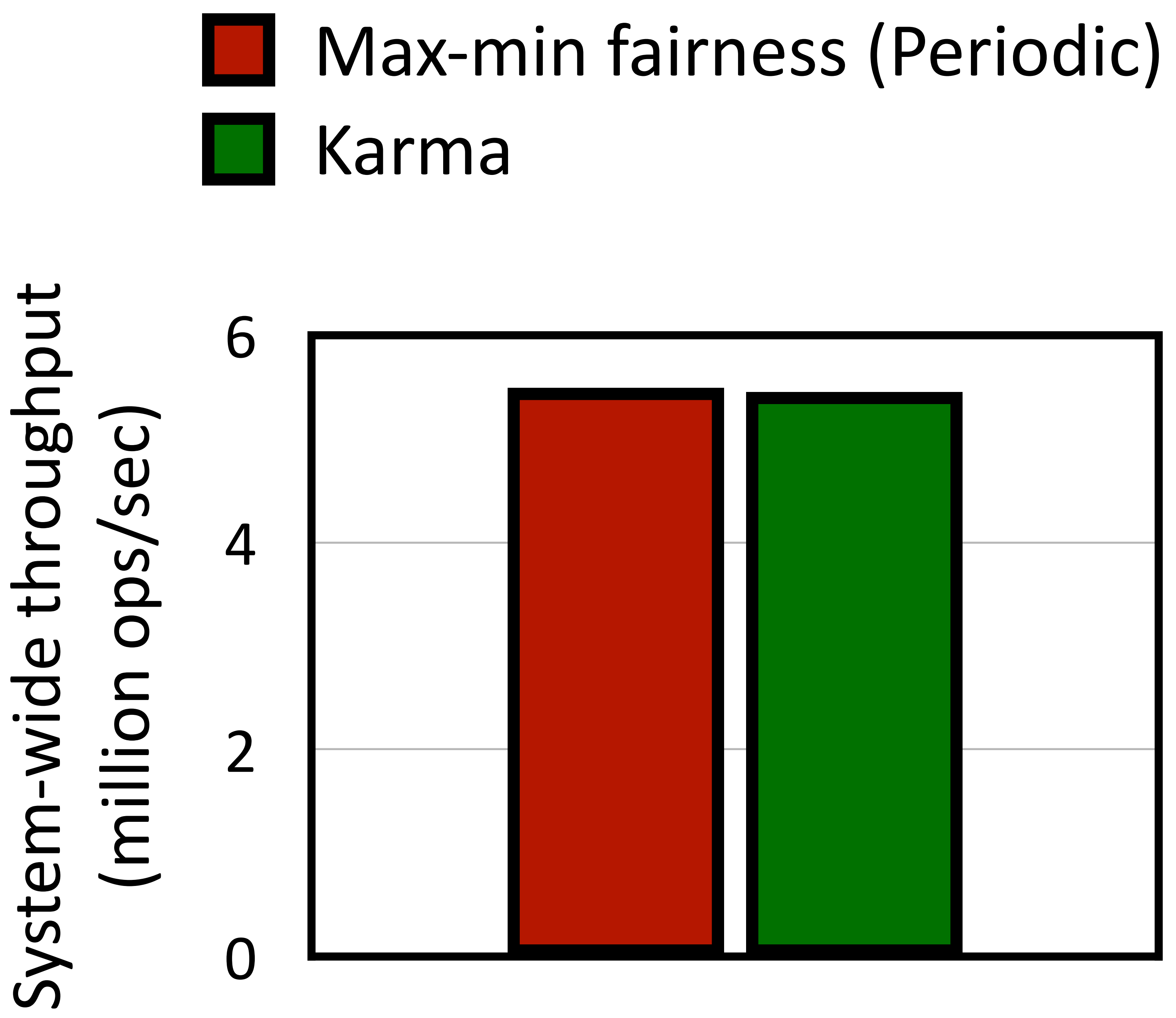
Glimpse of Karma evaluation results



Karma minimizes disparity across users
(while maintaining high average performance)

Glimpse of Karma evaluation results





Karma minimizes disparity across users
(while maintaining high average performance)

(See paper for more detailed evaluation results)

Karma: Revisiting the classical resource allocation problem under dynamic demands

New resource allocation algorithm for dynamic demands

Theoretical guarantees

- Pareto efficiency
- Strategy-proofness
- Fairness

Prototype implementation and evaluation

Karma: Revisiting the classical resource allocation problem under dynamic demands

New resource allocation algorithm for dynamic demands

Theoretical guarantees

- **Pareto efficiency**
- **Strategy-proofness**
- **Fairness**

Prototype implementation and evaluation

Karma opens many interesting avenues for future research, both in systems and in theory

Karma: Revisiting the classical resource allocation problem under dynamic demands

New resource allocation algorithm for dynamic demands

Theoretical guarantees

- **Pareto efficiency**
- **Strategy-proofness**
- **Fairness**

Prototype implementation and evaluation

Karma opens many interesting avenues for future research, both in systems and in theory

https://github.com/resource-disaggregation/karma

Midhul Vuppalapati (midhul@cs.cornell.edu)